PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

SEDE ESMERALDAS

CARRERA:

INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

TESIS DE GRADO

"DESARROLLO DE MICROSERVICIOS PARA LA NUBE UTILIZANDO ECLIPSE MICROPROFILE Y HEROKU"

LÍNEA DE INVESTIGACIÓN:

PROGRAMACIÓN Y DESARROLLO DE SOFTWARE

AUTOR:

BEDOYA BENAVIDES JAIR OSWALDO

ASESOR:

MSc. JAIME PAUL SAYAGO HEREDIA

ESMERALDAS, 2022

TRIBUNAL DE GRADUACIÓN

Trabajo de tesis aprobado luego de haber dado cumplimientos a los requisitos exigidos por el Reglamento de Grado de la PUCESE, precio a la obtención del titulo de Ingeniería en Sistemas y Computación.

Presidente del Tribunal de Graduación **f.**_____

Mgt. Jaime Sayago Heredia **f.**_____
Asesor

Mgt. Gustavo Chango **f.**_____
Lector #1

Mgt. José Luis Carvajal **f.**_____
Lector #2

Mgt. Susana Patiño **f.**_____
Coordinadora de carrera

AUTORÍA DE TESIS

Yo, **Bedoya Benavides Jair Oswaldo** con número de cédula de identidad 0803548015 manifiesto que mediante la presente investigación sobre el tema "**DESARROLLO DE MICROSERVICIOS PARA LA NUBE UTILIZANDO ECLIPSE MICROPROFILE Y HEROKU**" los resultados obtenidos como tesis de grado, previo a la obtención del título de "**INGENIERO EN SISTEMAS Y COMPUTACIÓN**" son de total responsabilidad del autor, y que se ha respetado las fuentes de información consultadas, realizando las citas correspondientes y los resultados alcanzados son totalmente personales, únicos y legítimos. Al mismo tiempo declaro que todo el contenido incluyendo resultados, discusión, conclusiones, recomendaciones y otros efectos legales y académicos que se desglosan, son y serán exclusiva responsabilidad legal y académica del autor y de la PUCESE.

_____

Bedoya Benavides Jair Oswaldo

C.I. 0803548015

DEDICATORIA

Este logro se lo dedico a Dios, por ser el pilar fundamental para conseguir y lograr tan anhelado sueño. Por brindarme salud y fuerza para salir adelante siempre en cada adversidad.

A mis padres que se esforzaron por darme una educación, mi hermano y mi familia entera que estuvieron en cada momento y cada paso que di en esta etapa universitaria. A mi abuelita y a mis tíos que me aconsejaron para poder salir adelante.

Jair Oswaldo Bedoya Benavides.

Agradecimientos

Agradezco a Dios por delante de todas las cosas, por otorgarme virtudes y bendiciones que permitieron poder cumplir con este objetivo en una etapa tan importante de mi vida.

A mis padres, por ser las personas que se esforzaron día a día, en cada año de estudios y haberme acompañado en este difícil camino siendo un apoyo fundamental en mi vida.

A los docentes y administrativos de la Escuela de Sistemas y Computación, quienes con mucho amor, dedicación y profesionalismo cumplieron con su principal labor de compartir sus conocimientos en esta etapa.

Jair Oswaldo Bedoya Benavides.

ÍNDICE

ÍNDICE DE FIGURAS

INDICE DE TABLAS

RESUMEN

En un principio, el enfoque principal de las organizaciones para implementar sus aplicaciones Java EE era empaquetar todos sus archivos, componentes y librerías en un solo archivo denominado EAR/WAR y llevar a cabo el archivo en un servidor dedicado para aplicaciones. Sin embargo, aunque este enfoque posee diferentes ventajas, desde la proyección de su fácil desarrollo, conlleva a una arquitectura monolítica, por lo que hace que las aplicaciones sean complicadas de mantener, difícil de corregir errores sin afectar toda la aplicación e imposibles de escalar en entornos basados en la nube (PaaS).

Los microservicios surgen como un modelo arquitectónico para brindar un sin número de ventajas, plantean y analizan las deficiencias que presenta la arquitectura. Estos microservicios a diferencia de empaquetar todos los microservicios en un solo archivo traen algunos beneficios como ser más flexibles, ser desarrollados de forma independiente y poder codificarse con varias tecnologías o con versiones recientes. Aunque la arquitectura de microservicios particularmente aborda toda problemática, poseen sus inconvenientes para el desarrollador como el conocimiento previo de implementar microservicios en diferentes contenedores requieren diferentes pasos, además de la configuración de estos, delimitar las dependencias, etc.

Por lo tanto, está investigación tuvo como propósito desarrollar una aplicación web conformada por microservicios en la nube mediante dos tecnológicas modernas como Eclipse Microprofile para el desarrollo de la aplicación en el lenguaje Java y Heroku para el despliegue de la aplicación en un entorno en la nube. Para el desarrollo de los microservicios se analizaron varios aspectos importantes como que framework y que versión utilizar, por lo que se utilizó el marco (framework) kumuluzEE y la versión 2.0 de Microprofile, se utilizó una base de datos Postgres para obtener, actualizar, crear y guardar datos.

Los resultados obtenidos demuestran la factibilidad y facilidad de desarrollar microservicios para un sistema de inventario implementado en la nube mediante la tecnología de Eclipse Microprofile que permitió crear los microservicios para obtener una aplicación autocontenida, delegando así funciones como la configuración de dependencias y despliegue, a su vez alta disponibilidad y escalabilidad.

Palabras Claves: Eclipse Microprofile, Microservicios, Arquitectura de microservicios, PaaS

ABSTRACT

Initially, the main approach for organizations to deploy their Java EE applications was to package all their files, components and libraries into a single file called EAR/WAR and run the archive on a dedicated application server. However, although this approach has several advantages, from the point of view of ease of development, it leads to a monolithic architecture, making applications difficult to maintain, difficult to fix bugs without affecting the entire application and impossible to scale in cloud-based environments (PaaS).

Microservices emerge as an architectural model to provide several advantages, raise and analyze the shortcomings of the architecture. These microservices, unlike packaging all microservices in a single file, bring some benefits of being more flexible, developing independently, change management and the use of various technologies or upgrading to recent versions. Although the microservices architecture particularly addresses all issues, it has its drawbacks for the developer as the prior knowledge of implementing microservices in different containers require different steps, in addition to the configuration of these, delimit dependencies, etc.

Therefore, the purpose of this research was to develop a web application consisting of microservices in the cloud using two modern technologies such as Eclipse Microprofile for the development of the application in Java language and Heroku for the deployment of the application in a cloud environment. For the development of the microservices several important aspects were analyzed such as which framework and which version to use, so the framework kumuluzEE and Microprofile version 2.0 were used, a Postgres database was used to obtain, update, create and save data.

The results obtained demonstrate the feasibility and ease of developing microservices for an inventory system implemented in the cloud using Eclipse Microprofile technology that allowed the creation of microservices to obtain a self-contained application, thus delegating functions such as the configuration of dependencies and deployment, as well as high availability and scalability.

Keywords : Eclipse Microprofile, Microservices, Arquitectura de microservicios, PaaS

INTRODUCCIÓN

Presentación de la investigación

En los últimos años, la computación en la nube (Cloud Computing) se ha definido como un modelo que ha sido utilizado de manera interactiva en las investigaciones educativas, científicas, industriales, así como también en los alojamientos de servicios y aplicaciones web. Siendo así, una plataforma que permite sostener modelos arquitectónicos de aplicaciones, servidores aplicativos y bases de datos [1].

Cloud Computing presta sus servicios mediante el internet, para poder distribuir el consumo de los servicios y aplicaciones web que permitan el uso de manera científica, empresarial e individual de manera segura [2].

Uno de los enfoques más populares que han surgido en los últimos años para desarrollar aplicaciones nativas en la nube ha sido la arquitectura de microservicios [3], su funcionalidad permite que las aplicaciones se desarrollen de manera autónoma y escalables en la interacción de sus servicios de modo que se desplieguen independientemente entre sí, realizando este proceso muy simple.

Los microservicios se encuentran definidos como un estilo de arquitectura en el desarrollo de una aplicación, que permiten acceder de una manera eficaz al uso de las herramientas disponibles en la nube que conlleva a estas tener propiedades más dinámicas y sutiles a diferencia de las aplicaciones tradicionales [4].

En este trabajo se presenta el desarrollo de microservicios como una arquitectura de gran impacto tecnológico en la actualidad con una gran importancia en la integración de aplicaciones web, con una alta escalabilidad y transaccionalidad. Para esto se detallan conceptos puntuales relacionados al desarrollo, implementación, despliegue en la nube y características principales en herramientas diseñadas para el desarrollo de esta arquitectura.

Los programadores utilizan diferentes herramientas que les facilite el desarrollo de microservicios debido a que es una arquitectura que permite a la aplicación web tener una mejor funcionalidad en todos sus procesos, por esto la investigación se enfoca en una herramienta en específico que permite desarrollar pequeños servicios o microservicios

.

Planteamiento del problema

En las últimas décadas, las grandes empresas como Netflix, eBay, Amazon, Twitter, PayPal ofrecen servicios relacionados con la web a millones de usuarios internautas que han aumentado de una manera repentina, por lo tanto, sus solicitudes e información debe ser controlables de manera segura y escalable.

En la actualidad las empresas quieren crecer no solo económicamente, sino que, de forma paralela en el ámbito tecnológico, con mayor seguridad, procesos agiles, mejor escalabilidad, mayor facilidad de implementación, innovación rápida y eficaz, agilidad al momento de realizar modificaciones, mayor escalabilidad y flexibilidad, aplicaciones más independientes para poder afrontar las nuevas exigencias que se presenten en un futuro.

Uno de los estilos de arquitectura que más ha sido utilizado es la arquitectura por capas o SOA (Arquitectura orientada a servicios), que en problemas sencillos de poca información han sido de gran ayuda, pero no para los nuevos retos existentes en la actual etapa de la información. Por esto, las grandes industrias decidieron dejar de utilizar su sistema tradicional y monolítico de las aplicaciones debido a que todo su sistema se localizaba en una sola aplicación. Muchas empresas al momento de observar estas desventajas que presentaban su arquitectura habitual decidieron migrar, actualizar e implementar aplicaciones basadas en la nube que manejen arquitecturas de microservicios [5].

Los últimos años la arquitectura basada en microservicios ha obtenido una buena aceptación por el motivo que facilitan que los servicios de las aplicaciones sean independiente y autónomas, permitiendo realizar cambios sin afectar la aplicación entera, además que puedan escalar de manera eficiente en reacción a los futuros requisitos, evitando resultados ineficientes [6].

Sin embargo, uno de los retos que posee esta arquitectura para los desarrolladores es entender y conocer cuando y como se debería usar, en un desarrollo sencillo no sería justificable invertir dedicación en dividir varias funcionalidades en microservicios, debido a que esto lentifica el desarrollo [7].

Por lo tanto, las preguntas de investigación que se plantean son: (i) ¿Es realmente eficaz desarrollar un microservicio para la nube en Eclipse Microprofile? (ii) ¿Cuáles son las principales herramientas para el desarrollo de microservicios en la nube? (iii) ¿Qué otras herramientas o framework serían utilizadas para poder realizar o desarrollar un microservicio alojado en la nube?

Justificación

La presente investigación tuvo como objetivo principal servir de guía técnica y práctica a los desarrolladores y empresas en la implementación de una arquitectura basada en microservicios con tecnologías en Java de tal modo que permitan diseñar una arquitectura descentralizada que pueda realizar cambios de forma rápida y sencilla en una aplicación que se requiera construir.

Al momento de desarrollar una aplicación web siempre se busca optar por una mejor arquitectura, al desarrollador se le hace difícil inclinarse por una tecnología que sea capaz de llevar a cabo un buen desempeño informático y de acoplarse a las necesidades requeridas, por lo que este trabajo de investigación va a permitir resolver que tecnologías y herramientas sean eficientes, innovadoras para el diseño de microservicios alojados en la nube.

Uno de los motivos que incentivó esta investigación fue saber que los desarrolladores pueden implementar microservicios desde varias tecnologías y herramientas de tal forma que puedan construir, implementar, mejorar y escalar partes de las aplicaciones de manera automática, debido a que estos microservicios trabajan de manera autónoma e independiente, siendo fácil su monitoreo y modificación de un servicio deficiente sin causar problemas en general.

Este trabajo de investigación aportó una demostración de cómo una arquitectura basada en microservicios alojadas en la nube desarrollada en ciertas tecnologías y varios lenguajes de programación, puede controlar varios procesos de manera automatizada e independiente para cumplir con las expectativas y problemas que se generan a medida que la aplicación web vaya generando más peticiones de los usuarios.

Objetivos

General

- Desarrollar dos microservicios mediante la utilización de herramientas como Eclipse Microprofile y Heroku que se comuniquen entre sí para ser utilizados en un sistema de inventario para la empresa Fujifilm.

Específicos

- Identificar los requerimientos funcionales y no funcionales para el desarrollo de los microservicios.
- Definir que marco (framework) de microservicio utilizar para el desarrollo del sistema.
- Desarrollar microservicios mediante el framework Eclipse Microprofile que permitan consumir y visualizar datos.
- Desplegar la aplicación web y alojarlo en la nube con la plataforma de Heroku.

CAPÍTULO I

1. MARCO TEÓRICO

1.1. Bases teóricas - científicas

1.1.1. Computación en la nube

Computación en la nube (Cloud Computing, en sus siglas en inglés) en los últimos años se ha vuelto tendencia refiriéndose a la entrega y dedicación con un enfoque en recursos informáticos y aplicaciones por medio de la web [8]. De la misma manera, facilita utilización de servicios debido a que está orientado a la escalabilidad, esto significa que puede encargarse de una solicitud muy fuerte que requiera de un servicio de una manera más directa e inmediata con relación al tiempo [9]. Estas características provocarán que los usuarios se den cuenta de que todo sistema o aplicación que se encuentre alojada en la nube tendrá un funcionamiento más completo, más rápido y sencillo por lo que los resultados serán más reconfortantes. Es fundamental enfatizar el requisito de una estandarización de servicios en la infraestructura, debido a que entre más estandarizada se encuentre, más fácil será todo su funcionamiento [10].

A pesar de que la computación en la nube no es una nueva tecnología se debe reconocer que es un modelo que más se destaca en la actualidad gracias a que ofrece servicios de poca demanda como almacenamiento, CPU, memoria, aplicaciones, servidores, ancho de banda de red, etc. y que permite distribuir los servicios de una infraestructura de manera más comprensible y a un nivel de exigencia mucho mayor [11]. Por otra parte, es un marco tecnológico y comercial, que permite eliminar la exigencia de mantener los hardware a causa de tecnologías como la virtualización en donde se agrupa todo en un mismo entorno [12]. Además, es un diseño de negocio conformado por varias cualidades que permitirán darle mayor interés a los beneficios y una gestión más rentable, reduciendo el consumo de los recursos de software y aumentando la optimización del costo de una infraestructura de tal forma que solo se paga lo que se consume mediante servidores virtuales en diferentes partes del mundo [13].

1.1.1.1.Tipos de nube

Los tipos de despliegues están clasificados y definidos por [14] en donde la computación en la nube puede ser implementada como pública, privada e híbrida según los requisitos que la empresa, organización o desarrollador requiera utilizar dentro de su infraestructura, servicio o aplicación.

Nube pública

Es un ambiente como su nombre lo dice público para un uso libre, en donde este entorno puede ser propio, administrado y operado por cualquier organización [15]. Los servicios ofrecidos por este tipo de nube se localizan en servidores no internos al cliente permitiendo ingresar sin restricciones a las aplicaciones [10].

Este tipo de nube posee ciertas ventajas como la suficiencia de procesar y almacenar, sin necesidad de la instalación de máquinas de forma local, evitando inversiones iniciales o gastos de mantener el equipo [13].

Sin embargo, también posee inconvenientes o desventajas como la disposición de toda la información de datos a empresas arbitrarias, así como también depender de sus servicios a través de Internet. Otras desventajas que se deben tener en cuenta es la baja seguridad que poseen en su infraestructura, es decir, son menos seguras que en relación con otros tipos de nubes, por otro lado, es su poca personalización que existe en ambos lados por parte del abastecedor de la nube y el cliente [13]

En este tipo de nube no es necesario de credenciales para conseguir paso a la información y servicios que la nube pueda brindar como se representa la Figura 1.

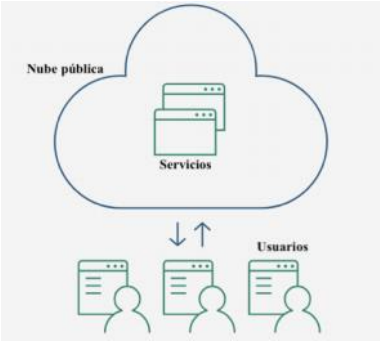


Figura 1.Tipo de nube pública

Nube privada

Son diseñadas específicamente para un solo usuario, empresa u organización, facilitando un control óptimo de la información que se vaya a gestionar, seguridad y atributo del servicio que se le ofreció al cliente [10]. Además, se enfocan en un entorno que se provisiona para el uso específico de una sola organización potencialmente que abarque múltiples grupos de consumidores [15]. El usuario en este tipo de nubes privadas es propietario de la infraestructura por la que fue diseñada la nube, teniendo un total control de las aplicaciones que son desplegadas dentro de ellas [13].

Una de las ventajas principales que posee este tipo de nubes es que los datos pueden ser localizados dentro de la propia organización o empresa, aunque pueden existir fuera de ellas, lo que genera a que exista una mayor seguridad dentro de estos sin intervención de terceras empresas o exposición a la población externa [10]. Otra ventaja es que existe un mayor control, privacidad y eficacia que proveen las nubes privadas [16].

Sin embargo, se presentan inconvenientes o desventajas en este tipo de nubes como la ampliación de sus sistemas informáticos, la escalabilidad se limita a los recursos otorgados por el servicio, los precios exagerados por poseer ciertas características adicionales y por su limitación a estar en un área definida [16].

En este tipo de nube los usuarios pueden ingresar a los servicios de la nube únicamente por medio de credenciales autorizadas como se observa en la Figura 2.

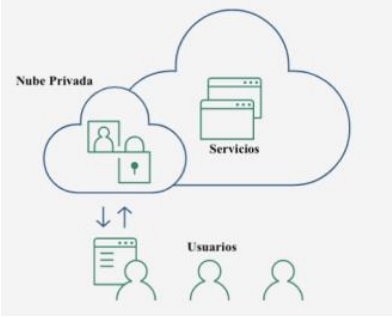


Figura 2. Tipo de nube privada

Nube híbrida

Es un entorno combinado entre dos o más infraestructuras de nubes distintas sean estas privadas o públicas de manera que se aprovecha las mejores características como la ubicación física de los datos que son manipulados por las nubes de una red interna privada con la simplicidad de gran parte de los recursos de las nubes públicas [10]. Sin embargo, se deben

tener ciertas precauciones como la confidencialidad y el resguardo de los datos, de la misma forma que se lo haría en la nube pública [13].

Muchas veces una parte de la información necesita ser confidencial con niveles de seguridad altos mientras que puede existir información que se necesite ser compartida entre grupos u organizaciones de acceso público por lo que trabajar en una nube pública sería lo recomendable como se ilustra en la Figura 3.

Figura 3. Tipo de nube híbrida

1.1.1.2.Modelos de servicios en la nube

Son varios los beneficios o capas que brinda la computación en la nube dependiendo de las necesidades y funcionamiento que los servicios requieran como se puede ilustrar en la Figura 4. Cada modelo tiene sus características esenciales que permitirán cumplir con las expectativas acorde a las necesidades que se necesiten utilizar para cualquier infraestructura. Además, cada capa está enfocada específicamente a grupos especializados en los servicios que brinde el modelo como se visualiza para IaaS están los arquitectos de Tecnología de la Información, asimismo para PaaS está enfocado hacia los desarrolladores y por último SaaS es adecuado para los usuarios finales, por lo que cada modelo será descrito a continuación.

Figura 4. Capas de los servicios ofrecidos en Cloud Computing

Infraestructura como un servicio (IaaS)

IaaS se define como un diseño de servicio en donde la parte del hardware se encuentra de forma virtual en la nube [10]. En este singular servicio, el proveedor brinda servicios tecnológicos (servidores, base de datos, redes), los cuales permiten romper con todos los moldes, debido a que se podría llevar a cabo desde un pequeño negocio hasta llegar a ser una gran empresa en el corto lapso. La aprobación de estos servicios está siendo popularizada debido a una gran parte de startups gracias a que iniciaron a emprender durante tiempos críticos en lo cual no se podía poseer un datacenter propio o una infraestructura definida [13].

Un ejemplo de IaaS es Amazon Web Services, en donde sus beneficios incorporan servicios de almacenamiento de datos, infraestructura enfocada a redes, máquinas virtuales siendo así servicios replicados a lo largo del mundo. El proveedor permitirá gestionar a los clientes sus sistemas dentro de un entorno virtualizado [10]. Este modelo de servicio está enfocado para cualquier organización que pretenda implantar sus aplicaciones y sistemas de software en un hosting o que requiera de la necesidad de usar servicios de almacenamiento, respaldos de seguridad o cálculos complejos de una amplia necesidad [13].

Entonces así, los que brindan servicios de internet son dueños de máquinas físicas mediante entornos que puedan ser gestionados, como por ejemplo una página web que permita llevar el control de inventarios de una empresa [10].

Plataforma como un servicio (PaaS)

Una PaaS se define como un modelo de servicio el cual está ubicado por delante de IaaS con relación al nivel abstracto que presenta en los bienes de servicios tecnológicos. Este servicio ofrece un entorno de software en donde el programador puede diseñar y personalizar respuestas mediante un conjunto de herramientas de desarrollo que proporciona directamente el servicio [9]. Este servicio puede estar enfocada en un lenguaje en singular o varios, así como frameworks de desarrollo. En este servicio de PaaS los usuarios pueden relacionarse con el software con la finalidad de encajar o recuperar datos y efectuar acciones sin tener la responsabilidad de mantener la parte del hardware, sino que únicamente interactuar con la plataforma [17].

En este tipo de modelo de servicio tiene como ideología que el programador de aplicaciones en la internet no se centra en almacenamiento de ficheros, gestión de la base de

datos, balanceo de máquinas, redes, escalabilidad, estabilidad, flexibilidad o modificar una maquina servidor [13]. El servicio sobre el que se despliega la aplicación web ya no es tarea del usuario, sino que es obligación del servicio o proveedor que se contrata y paga minuciosamente.

Las ventajas que brindan las plataformas como servicio son el enfoque total sobre la aplicación y el ahorro de costos, debido a que no solo se enfocan en vender u ofrecer almacenamiento y ancho de banda más económica, sino que lograr obtener el conocimiento para desplegar arquitecturas que escalando reflejan un gasto económico muy alto [10]. Sin embargo, existen las desventajas o problemas al depender de un único proveedor como el acoplamiento dentro del mismo y sus caídas que se presenten dentro de la plataforma que ofrece el proveedor [17].

Software como un servicio (SaaS)

Este servicio está posicionado en la capa más alta y se basa únicamente en la potestad que se le da al usuario de aprovechar las aplicaciones que se ejecutan y se encuentran alojadas en la nube [18]. Es decir, un modelo de despliegue de aplicaciones en donde el proveedor brinda licencias de su aplicación a los consumidores para su uso como un servicio de tal forma que pueda hacer uso dentro de la nube desde varios dispositivos sin manipular o controlar la infraestructura de la nube.

Este modelo de servicio SaaS ofrece la infraestructura, software y solución como un servicio global, además es el modelo más completo debido tiene la finalidad principal de brindar software y hardware a la misma vez, es decir un servicio en conjunto [17]. Además, se puede definir como un software que se encuentra alojado en un hosting y se puede acceder globalmente por medio del internet a través de un navegador, móvil, dispositivos digitales. En este modelo, los usuarios pagan por el uso que hacen por medio cuotas de registro, dentro de una etapa de tiempo de forma valida como si fuera una renta [13].

Las características principales que presenta en este modelo pueden ser resumidas en [13]:

- El programa se encuentra apto de forma global por medio de internet y bajo demanda.
- El diseño de subscripción por lo general se realiza por medio de licencias o apoyado en el consumo y es facturado mediante cuotas mensuales de forma recurrente.
- La parte operacional es obligación del que presta servicios de internet.
- SaaS permite soportar múltiples usuarios de forma multi-tenant.

Existen tres sistemas básicos que los proveedores desarrollan para poder habilitar SaaS como billing y facturación el cual es un potente sistema que permite soportar la suscripción por uso, subsistema clave el cual es el de autenticación, un sistema basado en single sign on para comunicarse con los sistemas CRM de las organizaciones para la habilitación del SaaS [13].

1.1.1.3.Ventajas, Desventajas y Proveedores de computación en la nube

La computación en la nube trae consigo una gran variedad de ventajas dentro de varios ámbitos como económicos, tecnológicos, ambientales y sociales de forma colectiva lo cual favorece de manera efectiva la consolidación dentro del mercado [19].

Una de las principales consecuencias al momento de acoger un nuevo modelo es que se deben asumir riesgos adicionales, todos estos riesgos son similares y comunes a los demás modelos que forman parte del Cloud Computing o sistemas de información es por esto que debe ser conveniente afrontarlos de frente y encontrar soluciones [13].

Los riesgos que presenta la computación en la nube son detallados a continuación:

- **Riesgo de pérdida de datos. –** A pesar de que tener los recursos desplegados en la nube y despreocuparse por la realización de backups diariamente, sin embargo, existe la amenaza de que la información en su totalidad se pueda perder incluso si se hace referencia de una nube pública. Existiendo una solución de escoger una nube hibrida, utilizando en la parte publica procesos con información menos importante.
- **Riesgo de caída del servicio. –** Una caída del servicio es otro riesgo mayor para cualquier uso que un usuario este realizando dentro de un servicio de la nube y más si es en un uso de IaaS pública. Afectando un tiempo determinado su funcionalidad y datos en diferentes plataformas como se puede observar en la Tabla 1

Tabla 1. Ejemplos de caídas de servicio

Servicio	Fecha	Duración
Amazon S3	15/02/2008	2 horas
Google Gmail	16/02/2008	30 horas
Ms-Azure	13/03/2009	22 horas
Salesforce.com	11/02/2009	6 horas
Microsoft Sidekick	4/10/2009	6 días + perdida de datos
Google Gmail	29/10/2009	2 horas
Amazon EC2	19/04/2011	14 horas

Las organizaciones, empresas o usuarios que se dedican a proveer los servicios que brinda la nube se los reconoce como Cloud Service Provider (CSP) (Proveedor de servicio en la nube, en español) y como se puede observar en la Tabla 2, los más reconocidos clasificados por modelos de nube son:

Tabla 2. Proveedores que ofrecen servicios en la nube

SaaS	PaaS	IaaS
Oracle On Demand	Microsoft Azure	Amazon Web Services
SalesForce	Heroku	Digital Ocean
AppDynamics	LunaCloud	IBM Cloud
Aprenda	Skyvia	CloudForge
Cloud Analytics	CloudForge	Dimension Data
Cumulux	Openstack	Linode
Intact	Amazon Elastic Beanstack	CloudSigma
NetSuite	Engine Yard	Linode

1.1.2. Arquitectura de microservicios

Las arquitecturas de microservicios fomentan el desarrollo de crear aplicaciones que sean mucho más complejas, en donde se forman de partes simples, independientes y autónomas, permitiendo trabajar en conjunto, pero de forma independiente cada servicio [20]. Está constituido de diversos servicios que ejecutan cada uno una labor diferente [7].

El enfoque principal de la arquitectura es distribuir un sistema en diferentes servicios sin que otros servicios sean afectados para tener una mejor optimización según los requerimientos del software, por lo que no sería necesario desarrollar una infraestructura con una complejidad tan alta [21].

A continuación, en la Figura 5 se observa la aplicación de microservicios RESTful en java desplegada en Oracle Cloud, cada microservicio se ejecuta en diferentes contenedores de Docker en un clúster albergado de Kubernetes utilizando además un balanceador de carga el cual selecciona que posible instancia es recomendable usar para procesar cualquier solicitud.

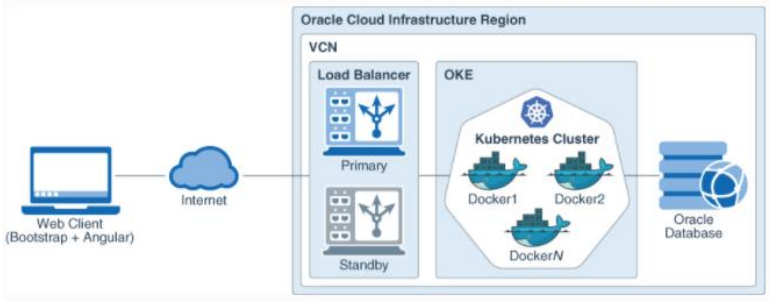


Figura 5. Arquitectura de microservicios en una aplicación

Describiendo la Figura 5, se pueden visualizar una aplicación compuesta por dos partes principales, demostrando como interactúa el front-end y back-end de microservicios y se obtienen ciertos aspectos como:

- La aplicación cliente de servicio web RESTful es desarrollada en HTML/CSS/JavaScript, es utilizada para acceder a la aplicación incluyendo los frameworks Bootstrap y Angular.
- El cliente se conecta a la aplicación mediante una red, siendo esta una red local en una máquina física o virtual.
- El back-end es un grupo de contenedores Docker que ejecutan servidores en un clúster de Kubernetes.
- La aplicación utiliza como base de datos una de Oracle para la persistencia, todos los datos recibidos se envían a esta y no se guarda ningún estado en el clúster del Kubernetes.

1.1.2.1. Microservicios

Es un enfoque de servicios pequeños que trabajan de forma conjunta para desarrollar una aplicación, ejecutándose de forma independiente en un único proceso [21]. Pueden ser desarrollados en diferentes lenguajes de programación y utilizar distintas tecnologías que permitan almacenar datos [21]. Además, permite acoplar componentes autónomos para efectuar las tareas que requieran [7]. Se podría definir también como servicios autónomos que trabajan a la par y son desplegados de forma individual y automatizada, utilizando una comunicación de recursos ligeros usando API (Interfaz de programación de aplicaciones) apoyados en HTTP/HTTPS, WebSockets o AMQP [21]. Los microservicios necesitan de un sistema de comunicación [22] entre ellos para poder realizar los procesos complejos de cada uno y pueden ser de tipo síncrono y asíncrono.

Al momento de desarrollar aplicaciones web esta arquitectura de microservicios ofrece la oportunidad de desarrollar sistemas divididos en unidades pequeñas que son autónomos e independientes, teniendo la capacidad de que una gran aplicación pueda ser presentada en un conjunto de aplicaciones [20]. Para poder abordar problemas complejos este nuevo estilo se impone en ambientes donde los requisitos de escalabilidad son bien estrictos y el desarrollo de los componentes separados sirven de gran ventaja.

El enfoque principal de la arquitectura de Microservicios es como dividen, aíslan y separan los servicios de un sistema o aplicación en contenedores diferentes como se observa en la Figura 6, en donde los servicios y las bases de datos se encuentran de forma independiente sin cada uno afectar en su proceso.

Figura 6. Arquitectura de Microservicios

Características de microservicios

Existen diversas maneras principales de caracterizar el uso de los microservicios

- Los microservicios son ideales para sistemas extensos, es decir que son aún más grandes que el tema o sistema a tratar ya que tienen problemas al momento de la edición. En otros términos, su exacerbación genera nuevos problemas
- Son perfectos para desarrollar aplicaciones de grandes escalas debido a que uno de sus retos principales es la alta disponibilidad. En contraste con la arquitectura monolítica que es adaptable en aplicaciones de poca transacción.
- El software desarrollado mediante microservicios puede descomponerse en diferentes partes funcionales e independientes. Cada servicio puede ser desplegado, modificado y vuelto a desplegar sin exponer otros servicios pertenecientes a la aplicación. Es decir, solo se tendrá que modificar los servicios que se requieran en lugar de redesplegar la aplicación entera nuevamente.
- Los microservicios son organizados acorde a sus necesidades, competencia, prioridades del usuario u organización en la que se integrara la aplicación. Se utilizan módulos multifuncionales, para que cada módulo se encargue de un servicio definido. El ahorra en relación con el tiempo de desarrollo es inmensurable además de la facilidad al momento de programar trabajos de mantenimiento, en donde se pueda supervisar los módulos mientras el resto de la aplicación no se ve interrumpida.

- La arquitectura de microservicios posee un diseño semejante al del gobierno descentralizado, dado que cualquier módulo cuenta con su propia base de datos a diferencia de recurrir todos a la misma saturándola de solicitudes y evitando a que si falla un servicio, toda la aplicación caiga.
- Los servicios pertenecientes a los microservicios están comunicados entre sí, por lo que cuentan como un sistema de notificación y actuación en caso de que algún servicio llega a fallar, depurando la información con destino al módulo perteneciente y realizando la correcta gestión de los servicios entre los demás módulos funcionales.

1.1.2.2. Comunicación en una arquitectura de Microservicios

Dentro de una arquitectura de microservicios, los servicios se efectúan en diferentes servidores por lo tanto debe existir una comunicación entre las diferentes instancias. La comunicación para enlazar estos servicios se genera por medio de protocolos diseñados como HTTP, AMQP y TCP. Uno de los protocolos para comunicar servicios son los mensajes HTTP/REST y asíncronos [23].

Para la comunicación de microservicios se definen modelos síncrono y asíncrono, en donde intervienen elementos de comunicación como uno a uno, el cual un mensaje se consume por un único consumidor y por otro lado el uno a muchos, un mensaje es consumido por múltiples consumidores como se puede observar en la Tabla 3.

Tabla 3. Mecanismos IPC

	Sincrónico	Asincrónico
Uno a Uno	Resquest/Response	Notificacion (Respuesta asincrónica)
Uno a muchos	----------	suscripción/publicación

Varios investigadores recomiendan usar el mecanismo IPC asíncronos que se orienten a eventos para poder comunicar entre servicios de tal forma que se utilice el modo de suscripción/publicación, donde los elementos pueden ser agregados y/o sustraídos sin causar un impacto grave en el software [24].

Por otro lado, los protocolos más frecuentes de comunicación en diferentes investigaciones y sistemas son a través de REST (Transferencia de estado representacional), como estándar con restricciones para los sistemas distribuidos, over HTTP o Thrift como mensajería síncrona y AMQP, en modo asíncrono el cual es un modelo que se orienta a

mensajes, encolamiento y enrutamiento de punto a punto como a suscripción/publicación a la altura de aplicación [24].

1.1.2.2.1. Tipos de comunicación

La comunicación entre el usuario y los diferentes microservicios puede efectuarse mediante varios tipos de comunicación, en el que cada uno se enfoca a un escenario distinto. Por lo tanto, existen dos criterios para clasificar estos sistemas de comunicación**:**

Por clase de protocolo:

- **Protocolo sincrónico. –** En este tipo de protocolo interactúan dos tipos de servicios que son, un usuario el cual remite un mensaje mientras que el servidor solo lo recibe. Sin embargo, la comunicación puede ser solo efectiva solo si el usuario recibe una respuesta por parte del servidor luego de procesar el mensaje. Este tipo de protocolo se caracteriza especialmente por ser un sistema que involucra aislar lo más posible cada uno del microservicio, debido a que se genera un bloqueo de los subprocesos. El http/https es un claro ejemplo de protocolo de comunicación entre microservicios de forma sincrónico, en donde el usuario puede seguir interactuando con sus tareas únicamente cuando el servidor le envíe una respuesta [7].

Un ejemplo como se observa en la Figura 7 sobre una comunicación sincrónica es una petición mediante el protocolo HPPT hacia una interfaz de programación de aplicación (API) como por ejemplo la de Spotify. Un usuario al efectuar una petición hacia la API para obtener datos sobre alguna música, por lo que el servicio que realiza esa petición tiene que aguardar que el servidor de Spotify remita una respuesta [7]


```
{
"artists": [
{
"id": "12Chz98pHFMPJEknJQMWvI",
"name": "Muse",
"type": "artist",
"uri": "spotify: artist:12Chz98pHFMPJEknJQMWvI"
}
],
"href": "https://api.spotify.com/v1/tracks/6KR9NaynH8bF9w727wKQBL",
"id": "6KR9NaynH8bF9w727wKQBL",
"name": "New Born",
"popularity": 47,
"type": "track",
"uri": "spotify: track:6KR9NaynH8bF9w727wKQBL"
}
```


Figura 7. Petición restclient – mode

- **Protocolo asincrónico. –** En este tipo de protocolo interactúan de la misma manera dos tipos servicios, para que la comunicación se realice de forma asíncrona el usuario emite la petición sin necesidad de esperar por una contestación, de esta forma el servidor percibirá la petición mientras que el usuario puede continuar efectuando una actividad diferente que tenga pendiente. Sus subprocesos no se encuentran bloqueados y usan protocolos compatibles con varios sistemas operativos y entornos dentro de la nube. El protocolo AMQP, trata sobre el código del cliente o el que envía el mensaje no espera ninguna respuesta a cambio, sino que solo envía el mensaje a una cola RabbitMQ o a cualquier otro tipo de agente de mensajes.

  Un ejemplo de comunicación asíncrona como se observa en la Figura 8, es a través del envió de un mensaje de texto en la aplicación de mensajería instantánea de Telegram mediante un bot que emite una petición a un usuario [7].

```
iex(1)> ExGram.send_message(14977303, "Hello")
{: ok,
%ExGram.Model.Message{
audio: nil,
text: "Hello", date:
1559203979,
chat: % {
first_name: "Cuwano", id:
14977303,
type: "private", username:
"Cuwano"
},
venue: nil,
reply_to_message: nil,
pinned_message: nil,
...,
from: % {
first_name: "Iron Test Bot", id: 376323488,
is_bot: true,
username: "irontest_bot"
}
}}
```

Figura 8. Bot enviando un mensaje a un usuario.

Por número de receptores:

Otro criterio de como clasificar los mecanismos de comunicación entre los microservicios con relación al número de receptores se clasifican entre un único o varios receptores:

- **Comunicación basada en mensajes de un solo receptor**

La comunicación basada en mensajes de un solo receptor hace referencia en que solo existe una comunicación punto a punto en el cual emite un mensaje a uno de los consumidores que hace uso de la aplicación y el mensaje se procesa una única vez. Este tipo de comunicación es

adecuada en especial para enviar comandos asincrónicos de un microservicio a otro como se puede observar en la Figura 9. Una vez que se empieza a utilizar la comunicación basada en mensajes se debe evitar juntar con la comunicación HTTP sincrónica [23].

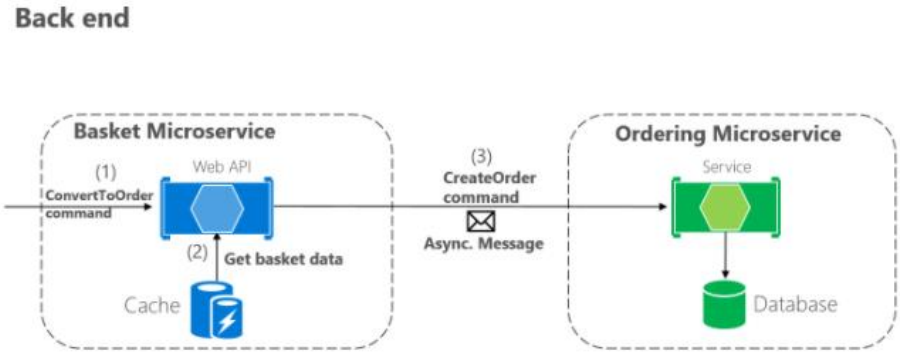


Figura 9. Comunicación de un solo receptor

1.1.2.2.2. Estilos de comunicación

A pesar de que los microservicios son independientes, esto no quiere decir que están aislados por lo que para comunicarse entre sí existen muchos protocolos que se pueden utilizar según el método de comunicación que se desee utilizar [24]. Si se utiliza un mecanismo de comunicación síncrono basado en solicitud/respuesta, los protocolos más comunes como enfoques son HTTP y REST, en especial si se encuentran sus servicios de forma pública fuera del hosting de Docker o del clúster perteneciente al microservicio. Sin embargo, si existe una comunicación interna entre servicios también se deben utilizar instrumento de comunicación en formato binario tales como WCF con TCP. De otra forma, se pueden utilizar mecanismos asíncronos basados en mensajes como AMQP [22].

- **Comunicación de solicitud / respuesta con HTTP y REST**

El estilo de comunicación de solicitud/respuesta se presenta cuando un usuario envía una solicitud a un servicio, la solicitud es procesada por el servicio y posteriormente envía una respuesta y es adecuada especialmente para consulta de datos en tiempo real mediante una interfaz de usuario desde aplicaciones de tipo cliente. Por lo tanto, este mecanismo será utilizado para la mayoría de las solicitudes en una arquitectura de microservicios lo que se puede visualizar en la Figura 10.

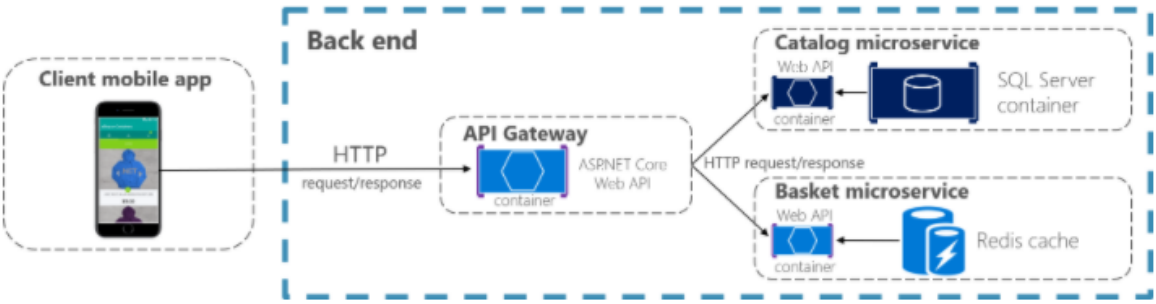


Figura 10. Uso de comunicación de solicitud/respuesta HTTP

Este tipo de comunicación se usa cuando el cliente asume que la respuesta le llegara pronto, por lo general menos de un segundo, o a lo mejor unos pocos segundos como máximo. En caso de no obtener lo requerido se aplican otras medidas como patrones y tecnologías de mensajería siendo un enfoque diferente a los tipos de comunicación pero que permitirán tener resultados más rápidos y eficientes [24]. Un estilo arquitectónico que se usa de forma popular para este mecanismo es el REST siento el enfoque más utilizado al crear servicios, además se basa en el protocolo HTTP abarcando verbos como GET, POST y PUT [25].

- **Comunicación push y en tiempo real basada en HTTP**

La comunicación HTTP en tiempo real hace referencia que puede hacer que el código dentro del servidor remita contenido a los usuarios conectados en consecuencia que los datos se encuentran disponibles, a diferencia de que el servidor espere a que un usuario solicite datos nueves como muestra la Figura 11 [22].

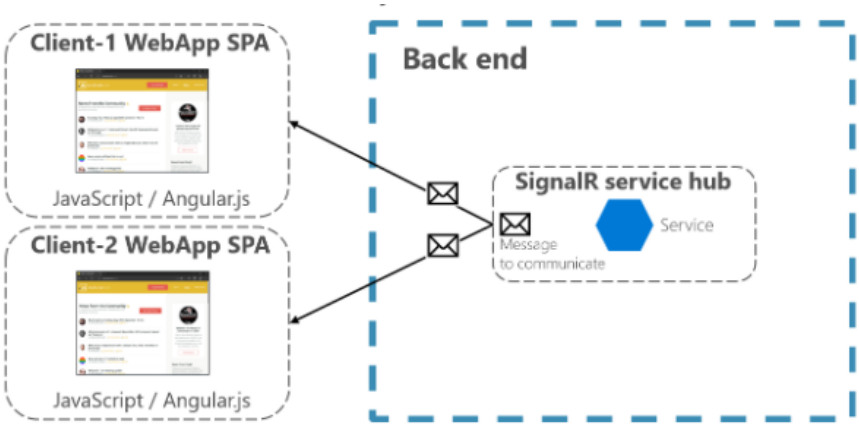


Figura 11. Comunicación de mensajes asíncronos de uno a muchos.

En la Figura 11 hace uso de SignalR la cual es una excelente forma de poder obtener una buena comunicación en tiempo real para poder emitir contenido a los usuarios desde un servidor back-end debido a que la comunicación se efectuara en tiempo real, en donde los cambios en la aplicación se muestran casi al instante. Por lo general, se supervisa por medio de un protocolo como WebSockets utilizando conexiones por cada cliente.

1.1.2.3.Ventajas y Desventajas

Los microservicios con todas sus características generan ventajas como [7]:

- Software más sostenible, es decir al poseer servicios pequeños son mucho más fáciles de comprender y, si fuera el caso, de modificar.
- Simple de experimentar, al tener funcionalidades concretas comprimidas en un servicio reducido es más fácil diseñar test para supervisar su correcto funcionamiento.
- Factible de desplegar, diferente a la arquitectura monolítica no es necesario volver a desplegar la aplicación en conjunto, suficiente con detener y arrancar el servicio que se requiera modificar o reparar.
- Concede la estructura del desarrollo en equipos autónomos e independientes.
- Ofrece a los desarrolladores la libre expresión de programar y desplegar los servicios de manera independiente.
- Cada módulo puede tener o usar diferentes lenguajes de programación y con tecnologías más actuales.

Sin embargo, de la misma forma que otras arquitecturas, los microservicios poseen diversas complicaciones como [7]:

- La gran mayoría de los DevOps de microservicios necesitan tratar en base a la comunicación que requieren los microservicios.
- Generar test para verificar la interacción entre los servicios es dificultoso.
- Al momento de desplegar una aplicación resulta ser muy complicado si existen demasiadas dependencias en los servicios, debido a que deberían ser desplegados en el orden correcto por lo cual es una desventaja para un despliegue sencillo.
- Al momento de tener una comunicación entre los servicios, puede generarse que el sistema se vuelva impredecible y con una gran complejidad de ser entendible.

1.1.2.4.Contenerización

Un contenedor puede ser definido como “objeto de almacenar, asegurar o transferir algo”, por lo que un contenedor de software es semejante, además tienen solo lo que se necesita para desplegar la aplicación [26].

A diferencia de un entorno virtual, este posee por completo un sistema operativo por lo que ocupa demasiados recursos en la memoria del procesador produciendo que el sistema arranque un poco más lento y sin tener la posibilidad de tener una variedad incontable de máquinas en ejecución, a diferencia de los contenedores pueden tener un sin número de servicios siempre y cuando se tenga suficiente hardware como se representa en Figura 12 [25].

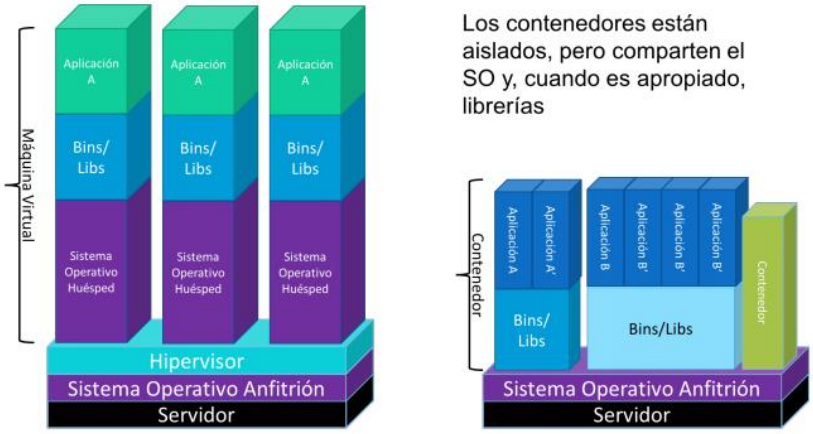


Figura 12. Recursos de entornos virtuales y contenedores

Los contenedores carecen de núcleos del sistema operativo lo que permite que sean muchos más rápidos y flexibles, además puede dirigir cada aplicación en varios procesos con múltiples conjuntos separados [25].

Sin embargo, tener una máquina virtual permite alojar varios sistemas operativos huésped a diferencia de los contenedores, pero aún esta desventaja no los hace menos operacionales por el motivo que de comparten sus sistemas operativos cuando se requiere de forma inalámbrica y utilizando las librerías apropiadas.

1.1.2.5.Arquitectura Monolítica vs Microservicios

La arquitectura de software con el pasar de los años ha sufrido una transformación increíble en relación a la reducción de tiempo y costos, creación de nuevas arquitecturas que permitan que los procesos sean mucho más eficientes [8].

Una arquitectura monolítica es una única unidad en donde las aplicaciones son desarrolladas en tres partes diferentes como el frontend, backend y base de datos mientras que la arquitectura de microservicios es una agrupación de pequeños y simples servicios, en donde se ejecutan de forma independiente comunicándose mediante una API http o REST ya sea de forma síncrona o asíncrona como se observa en la Figura 13.

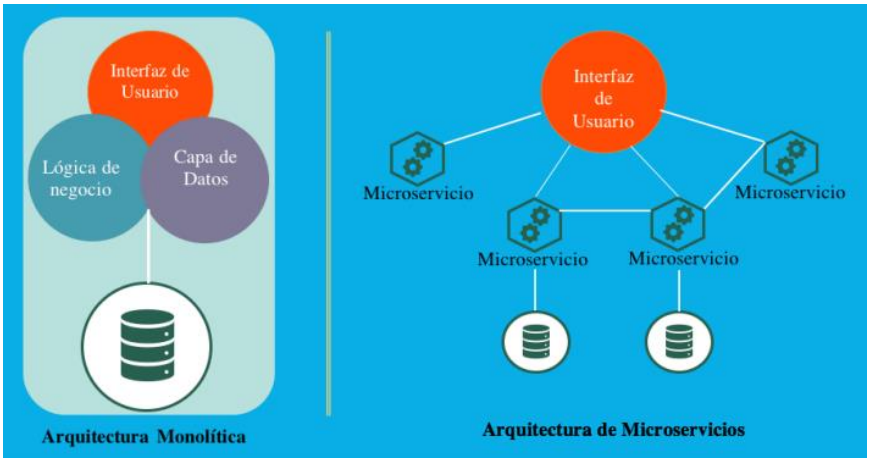


Figura 13. Arquitectura monolítica y de microservicios

Características entre arquitectura monolítica y de microservicios

Ambas arquitecturas presentan características principales en el mismo aspecto lo que se define como una comparación importante como se puede visualizar en la Tabla 4 [8]:

Tabla 4. Características entre arquitectura monolítica y de microservicios

Tamaño: En la arquitectura monolítica su gran tamaño y complejidad afecta de forma negativa en el mantenimiento a diferencia que en la arquitectura de microservicios se encuentra implementado en una cantidad limitada de pequeñas funcionalidades lo cual reduce el porcentaje de errores.
Dependencias: En las aplicaciones monolíticas se presenta una exigente dependencia entre el código que conforma el sistema, en las aplicaciones con microservicios se genera la posibilidad de poder migrar a nuevas modificaciones de diferentes tecnologías.
Actualización: En los sistemas monolíticos modificar o actualizar un módulo requiere el reinicio de toda la aplicación o sistema por completo, mientras que, el manejo modular en los microservicios permite evadir que se reinicie por completo el sistema, y solo modificar la parte afectada.

Despliegue: Las arquitecturas monolíticas son mucho más complicado que un despliegue tenga éxito por completo debido a la cantidad de requerimiento de los recursos, por otra parte, los microservicios tienen un enfoque principal que les permite desplegar en un ambiente favorable y compartido gracias al uso de contenedores.
Escalabilidad: El sistema monolítico por lo general presenta un límite en la escalabilidad a diferencia que el sistema de microservicios la escalabilidad no es un riesgo alguno, puede hacer frente a cualquier fallo que se presente.

En la siguiente Tabla 5 se resumen las diferencias entre los microservicios y las arquitecturas monolíticas, haciendo relación en varios aspectos en relación cuando la aplicación o el software sea desarrollado y desplegado.

Tabla 5. Arquitecturas monolíticas vs Microservicios

Categoría	Arquitectura Monolítica	Microservicios
Código	Está formado de un solo código para todo el software.	Están desarrollado por parte con su propia base de código.
Comprensibilidad	Esdifícil de entender y mantener.	Fácil de comprender.
Despliegue	Cualquier cambio o modificación requiere de redespliegue y reinicio total de la aplicación.	Pueden ser desplegados de manera más sencilla y de forma independiente.
Lenguaje	Se programa en un solo lenguaje de programación.	Los microservicios pueden ser programados en diferentes lenguajes de programación.
Escalabilidad	Es indispensable realizar un escalamiento a la aplicación entera.	Pueden ser escalados de forma independiente sin afectar los demás servicios.
Comunicación	Existen funciones o procedimientos que simplifican la comunicación entre los componentes de la aplicación.	Utilizan el modelo de comunicación de solicitud-respuesta. Llamadas a la API de REST basadas en el protocolo HTTP
Mantenimiento	En relación con que aumenta los procesos de la aplicación, el mantenimiento del código se vuelve mucho más complejo	La aplicación es más fácil de mantener debido a que los microservicios son independientes y sencillos.

En el artículo [27], titulado “The evolution of distributed systems towards microservices architecture”, se realizó un análisis profundo con relación a cuatro sistemas distribuidos listando las características que estos sistemas presentaban y así poder realizar una comparación descriptiva entre sus tipos de arquitecturas como se ilustra en la Tabla 6.

Tabla 6. Comparación característica de Arquitecturas

Características	Arquitectura Monolítica			Arquitectura de Microservicios
	Cliente/ Servidor	Agentes móviles	SOA	
Red	NO	SI	SI	SI
Comunicación en procesos internos	NO	SI	NO	SI
Escalabilidad elástica	NO	NO	NO	SI
Comunicación ligera	NO	NO	SI	SI
Resistencia	NO	NO	SI	SI
Rapidez en entrega de software	NO	NO	SI	SI
Integración de servicio autónomo	NO	NO	SI	SI
Servicio potable	NO	SI	SI	SI
Reusabilidad de servicios	NO	NO	SI	SI
Migración de servicios	NO	SI	NO	SI
Servicios Monolíticos	SI	SI	SI	NO
Perdida de acoplamiento	NO	NO	SI	SI
Basado en la nube	NO	NO	NO	SI
Desarrollo multifuncional	NO	NO	SI	SI
Separación funcional	NO	NO	SI	SI
Middleware	NO	NO	SI	NO
Tolerancia a fallos	NO	NO	NO	SI
Manejo descentralizado	NO	NO	NO	SI
Bajo costo de recursos	NO	NO	NO	SI
Servicio independiente de despliegue	NO	NO	NO	SI

Además, en la Tabla 6 se puede deducir a través de los resultados las impactantes limitaciones que las arquitecturas monolíticas exhiben, por lo que se presenta una incertidumbre por desarrollar un software funcional válida para cualquier dispositivo o aplicación. En la actualidad lo que predomina son los dispositivos inteligentes, debido a que es mucho más factible cargar un aparato manejable, pequeño y cómodo que permita usar los servicios o aplicaciones que antes solo se podían utilizar mediante una portátil o un computador de mesa. Con este ejemplo, se realiza una comparación sobre tratar de seguir desarrollando software con arquitectura monolítica lo cual podría generar problemas para el desarrollador como pérdida de tiempo y costos muy altos [8], [27]

1.1.3. Patrones de diseño

Hay dos formas de diseñar e interactuar una arquitectura, diseñar una arquitectura desde cero o buscar resoluciones ya propuestas a nuestro problema, si escogimos la segunda opción, hay patrones de diseño, patrones de arquitectura y framework [28].

Los patrones representan un método reutilizable para resolver problemas comunes, los primeros definidos en arquitectura civil se pueden encontrar en el libro "A Pattern Language" [29], que describe los patrones con un nombre de tal forma que al referirse a ellos cualquier arquitecto se capaz de entender qué solución se utiliza sin entrar en detalles, pero en arquitectura de software no aparecieron hasta 1994 en un libro que sigue siendo un modelo de referencia en la actualidad.

Los patrones definidos en el libro [29] se parten en funcionalidad del problema a solucionar:

PATRONES DE CREACIÓN	PATRONES DE ESTRUCTURA	PATRONES DE COMPORTAMIENTO
Se utilizan para facilitar la creación de nuevos objetos. Los más usuales son: • Abstract Factory • Factory Method • Builder • Singleton • Prototype	Dichos nos facilitan la modelización de nuestra aplicación, explicando la manera en que las clases se relacionan entre sí. Los más usuales son: • Adapter • Bridge • Composite • Decorator • Facade • Flyweight • Proxy	Se utilizan para gestionar algoritmos, relaciones y responsabilidades entre objetos, siendo las más conocidas • Command • Chain of responsibility • Interpreter • Iterator • Mediator • Memento • Observer • State • Strategy • Template Method • Visitor

1.1.4. Patrones de microservicios

Un patrón de microservicios está integrado por una serie de conceptos, que tienen como objetivo buscar solución a problemáticas que se presenten en un área específica de la aplicación desarrollada mediante la arquitectura de microservicios, permitiendo que la

aplicación funcione de una manera totalmente independiente, desacoplada en su totalidad y pueda ser escalable y tolerante a fallos [30]–[32].

Existen diferentes patrones de microservicios que permiten cumplir los requerimientos presentados y se clasifican de la siguiente manera.

Patrón de microservicios de agregador

El patrón de agregador convoca múltiples servicios para poner en marcha las funciones solicitadas por la aplicación. Cuando un usuario necesita información que se encuentran en diferentes microservicios este patrón invoca un microservicio que permite agregar las llamadas a diferentes microservicios y así obtener una respuesta [31]. Este tipo de diseño puede ser aplicado en una página web sencilla que procese y muestre los datos recuperados, en donde luego de agregar la lógica empresarial a los datos que serán recuperados, se independiza en un nuevo microservicio teniendo cada servicio su propia caché y base de datos y si el patrón o nuevo microservicio es un servicio compuesto, este también poseerá su caché y base de datos propios. Por lo tanto, este patrón es el más utilizado y simple como se puede observar en la figura

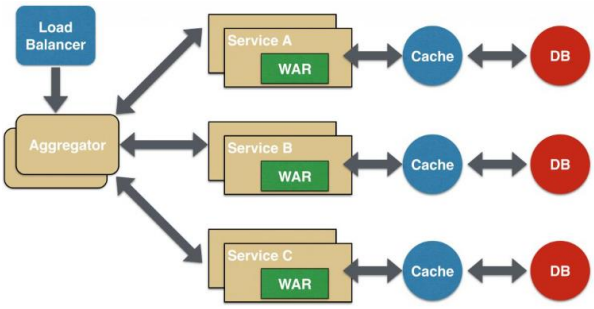


Figura 14. Patrón agregador

Patrón de microservicios del agente

A diferencia del patrón de agregador, este presenta una modificación en donde el usuario no adiciona datos, sino que más bien invoca a varios microservicios acorde a la diferencia en los requisitos que se presenten, por lo tanto, el patrón del agente solo puede comisionar o encargar solicitudes o efectuar funciones de conversión de datos, pero con las mismas características del patrón de agregador, pero con objetivos similares [30].

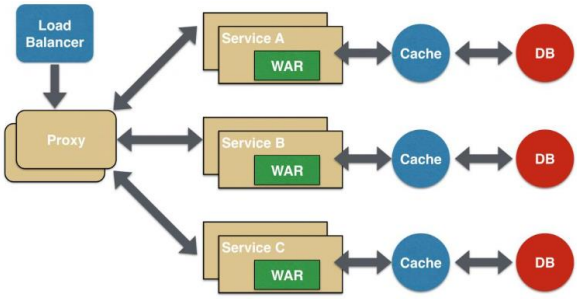


Figura 15. Patrón del agente

Patrón de microservicios de cadena

El patrón de cadena permite realizar llamadas consecutivas o de seguimiento para tener una composición efectiva en base a la información, los servicios se comunicarán entre si después de recibir la solicitud utilizando una mensajería sincrónica por lo que esta cadena no debe ser muy extensa para así evitar que el cliente tenga que esperar mucho tiempo la solicitud [30]. De igual manera que los anteriores patrones, cada servicio posee su propia cache y base de datos.

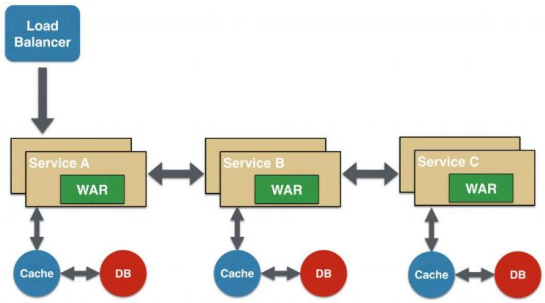


Figura 16. Patrón de cadena

Patrón de microservicios de rama

El patrón de rama es una extensión del patrón agregador con la diferencia que permite invocar a dos cadenas de microservicios en el mismo momento hacia diferentes microservicios o servicios que poseen su propia base de datos y caché almacenada [30].

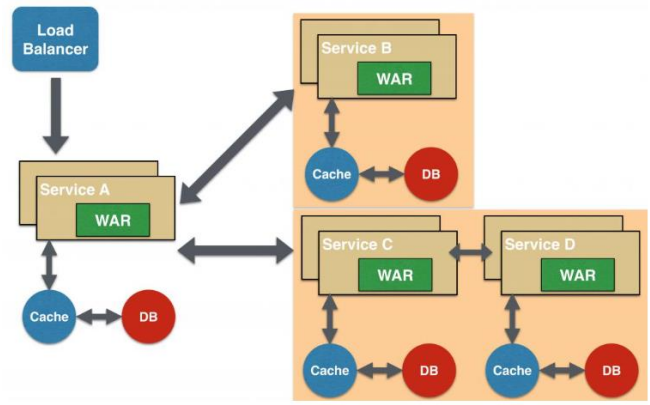


Figura 17. Patrón de rama

Patrón de microservicios de mensajería asincrónica

Este patrón brinda comunicación asincrónica a través de diferentes microservicios por medio de un modelo de sondeo asincrónico. Es decir, cuando la parte del back-end obtiene una solicitud o llamada, este responde inmediatamente por medio de un identificador de solicitud y, procede a llevar la solicitud a cabo de forma asincrónica [31]. Este tipo de patrón evita el cuello de botella que es causado por la comunicación sincrónica. Sin embargo, una desventaja común de este patrón suele ser que las transacciones comerciales son sincrónicas de acorde a su funcionamiento de la aplicación.

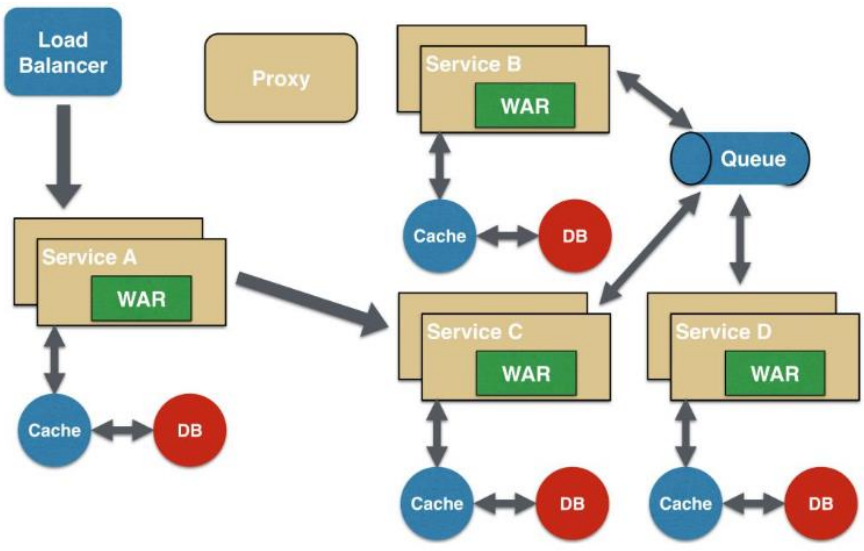


Figura 18. Patrón de mensajería asincrónica

1.1.5. Seguridad para microservicios

Los microservicios posibilitan separar grandes aplicaciones monolíticas en servicios implementados de forma independientes, con la desventaja que estos servicios desacoplados aumentan el número de elementos, y surge la dificultad y complicación de hacer segura la aplicación [33].

Una infraestructura común de microservicios posee diferentes capas de comunicación, virtualización, contenerización, orquestación y servicios por lo que cada uno tendrá requisitos y desafíos de seguridad que cumplir. Dando por entendido que entre más servicios de forma independiente existan, mayor será el desafío de seguridad.

Las alternativas más frecuentes que implican RESTful y seguridad de microservicios se relacionan en los estándares Open ID Connect (OIDC), Oauth2 y JWT.

Los token de seguridad de tipo JWT brindan configuración liviana e interoperable de difundir identidades mediante diferentes servicios, en donde estos no necesitan almacenar ningún estado sobre clientes o usuarios, Pueden examinar y comprobar el token generado de forma local si se encuentra en formato JWT o si se encuentra de forma remota realizado con un proveedor de confianza, además pueden determinar al cliente que realiza la petición y comprobar que el servicio definido sea suceso de una supuesta solicitud del toque [33].

- Los servicios pueden atribuir políticas que permiten autorización en relación con información general token.
- Los servicios pueden emplear el token generado tanto para la representación como para la sustitución de identidades.
- Existe compatibilidad con la delegación y la suplantación de identidad es sobre clientes.
- Los servicios no requieren almacenamiento de ningún tipo de estado sobre estos clientes o usuarios.
- Los servicios pueden examinar la autenticidad del token solo si el token sigue un formato conocido caso contrario estos servicios pueden increpar un servicio distinto.

Autenticación basada en JWT

Es un formato de representación de solicitudes diseñadas para entornos con invitaciones faciales como HTTP que consta de encabezados de autorización, autenticación y parámetros de consulta URI [34]. Los JWT codifican solicitudes para ser transferidos como un objeto JSON que se usa como una carga útil referente a una estructura Jason web signatura o como un texto sin formato mediante una estructura JSON Web Encryption, lo que facilita las peticiones de estar firmadas de manera digital o de respaldar la integridad como un mensaje

de código de autenticación (MAC). Los JWT son siempre interpretados utilizando una serialización compacta JWS o JWE.

Autenticación basada en token

Los protocolos de autenticación referentes a seguridad basados en token autorizan que los sistemas autentiquen, permitan y verifiquen identidades en relación o función de un Token de seguridad [34]. En lo común, los siguientes atributos se encuentran involucrados en este tipo de protocolos:

- **Emisor**: Es el responsable de manifestar diferentes Token de seguridad como efecto de comprobar con éxito una autenticación o identidad. Por lo general, suele existir una relación entre los emisores y los proveedores de identidad.
- **Cliente**: Es la representación que realiza una aplicación para la que se generó el token. Estos clientes en lo común se encuentran relacionados con los proveedores de servicios. Un cliente puede ejercer como intermediario entre un servicio o delegación y un sujeto.
- **Sujeto**: Es la entidad o atributo a la que hace referencia la información en un token.
- **Servidor de recursos:** Se representa mediante una aplicación para así verificar acceso a materiales protegidos.

Pasos para la autenticación del token

1) **Extrae el toque de seguridad de la solicitud**

Para que los servicios restFul por lo general se logra obteniendo el token del encabezado de la autorización correspondiente [34].

2) **Realiza verificaciones de validación contra el token**

En este paso por lo general depende de qué formato tenga el token y del protocolo de seguridad que se está usando. El principal objetivo es estar seguros de que el token sea válido y permita ser consumido mediante la aplicación. Puede involucrar validación de firmas, cifrado y caducidad [34].

3) **Introspección de token y extracción de información sobre el tema**

Por lo general en este paso depende de qué formato tenga el token y el protocolo de seguridad que se está usando. El principal objetivo es recaudar toda la información imprescindible sobre el tema referente al toque [34].

4) **Crear un contexto de seguridad para el sujeto**

En relación de la información recaudada mediante el token, la app genera un formato de seguridad para el sujeto con la finalidad de emplear la información donde sea necesario al facilitar los servicios los recursos protegidos [34].

Beneficios de JWT

- La validación del toque no exige una petición adicional y permite ser válida de forma local por cada servicio [34].
- Dado su formato JDON, este se basa únicamente en peticiones o atributos para transportar información de autenticación autorización y comprobación sobre un cliente [34].
- Proporciona el soporte de varios tipos de mecanismos que permitan controlar el acceso como ABAC, RBAC, control de acceso basado en contexto, entre otros [34].
- Estabilidad a la altura del mensaje mediante una firma y encriptación según lo generado por los estándares JWS y JWE [34].
- Las partes involucradas pueden llegar cortar fácilmente un conjunto específico de solicitudes para intercambiar información de autorización y autenticación [34].
- Las principales tareas de la especificación MP-JWT, son la definición junto con la API de Java y la asignación a las API de JAX-RS [34].

Figura 19. Ciclo de vida de un token JWT

Autenticación Basada en token mediante Open ID Connect

Es un protocolo de seguridad especializada en identidad basado en OAuth 2.0 [35]. Este protocolo posibilita que los clientes puedan o tengan autorización de verificar la identidad del usuario en relación con autorización generada mediante un servidor autorizado, así como también extraer información básica del usuario de forma interoperable e igual a RESTful.

Open ID Connect posibilita a todos los clientes sean de tipo web, móvil y Java script, requerir y acoger informaciones referentes a sesiones y usuarios autenticados [35].
Este protocolo contiene un conjunto de especificaciones muy amplias, permitiendo a los integrantes emplear servicios opcionales, como el cifrado de datos, gestión de inicio de sesiones y proveedores de Open ID.

Autenticación Basada en token mediante OAuth 2.0

Este protocolo de seguridad posibilita y autoriza una aplicación de terceros adquirir un acceso limitado a un servicio HTTP [36], siendo el nombre del propietario de un recurso, orquestando una interacción de aceptación entre el propietario y servicio de recursos y como también el servicio HTTP consecutivamente, de manera que la aplicación tenga autorización para conseguir acceso en su propio nombre.

Diferencia entre OpenID y Oauth

OpenID es un modelo de seguridad para la autenticación mientras tanto que OAuth es para autorización [35]. La autenticación hablamos de aseverarse de que el individuo con la que estás hablando es, por cierto, quien dice ser. La autorización hablamos de dictaminar qué se le debería permitir a aquel tipo.

En OpenID, la autenticación está encargada: el primer servidor desea autenticar al primer cliente, sin embargo, las credenciales del primer cliente (por ejemplo, el nombre y la contraseña de primer usuario) se envían al otro segundo servidor, en el cual el primer servidor confía (al menos, confía para autenticar usuarios). Por cierto, el segundo servidor se garantiza de que el primer cliente sea verdaderamente aquel cliente, y después le da la aseveración al primer servidor. Mientras que, En OAuth, la autorización es encargada: la primera entidad recibe de la segunda entidad un "derecho de ingreso" que la primera entidad puede demostrar al primer servidor para que se le otorgue ingreso; Por consiguiente, la segunda entidad puede dar claves de ingreso temporales y concretas a la primera entidad sin darles bastante poder.

1.1.6. Aplicaciones para desarrollo de microservicios

1.1.6.1. Eclipse Microprofile

Eclipse Microprofile es un entorno para desarrollar aplicaciones web fundamentadas en microservicios exclusivamente para desarrolladores de Java. También se menciona que es una especificación de software libre para el desarrollo y uso de microservicios pertenecientes a Java Enterprise, teniendo un objetivo principal como el de optimizar un ambiente descentralizado y enfocado en una arquitectura de microservicios [37]. Posee también una serie de especificaciones implementadas en cada versión como configuración, tolerancia a fallos, JWTs (JSON Web Tokens), metricas, comprobaciones de salud, propagación de JWT, OpenTracing, OpenAPI y el REST siendo capaces de dar posibilitar la ejecución de microservicios desplegados en la nube. A continuación, sus especificaciones:

- **Configuración:** Es la capacidad que poseen las aplicaciones para poder ser configuradas en relación con el entorno de ejecución. Utilizan archivos de diferentes formatos con la finalidad de contar con la posibilidad de emplear una actualización en las propiedades de configuración sin tener la obligación de volver a construir y empaquetar nuevamente la aplicación. Por lo que en microservicios que son ejecutados en la nube la especificación Microprofile Config es un requisito extremadamente importante [37].
- **Cliente Rest:** Este tipo de especificación, determina como increpar servicios de tipo RESTful mediante protocolo HTTP [37].
- **Tolerancia a fallos:** La resistencia o tolerancia a fallos es una de las principales características más importantes que son requeridas dentro de los entornos de la nube para los microservicios. Define una manera de proporcionar una variedad de estrategias que permitan llevar a cabo la ejecución y el resultado implementado por el código [37].
- **Métricas:** Las métricas son muy esenciales recuperar en entornos de ejecución y dentro de un entorno distribuido como PaaS. Es una forma estándar que ayuda a que los servidores de MicroProfile exhiban los datos de monitorización a los encargados de gestión y utilizar para la exposición de sus datos telemétricos a los equipos operacionales [37].
- **OpenAPI:** Esta especificación tiene como objetivo principal definir una API Java que permita implementar la especificación de OpenAPI en su respectiva versión [37].

- **Propagación de JWT:** La especificación permite garantizar la seguridad de las APIs expuestas recreando en los sistemas la validación de autenticación y autorización de cada petición. Entre los principales protocolos de seguridad se encuentran: OAuth2, WS-Federation, OpenID Connect, WS-Trust y Security Assertion Markup Language y basándose en estándares definidos por OAuth2, JWT y OpenID Connect para la propagación de JWT [37].
- **OpenTracing:** Los microservicios están constantemente ejecutándose en entornos de sistemas distribuidos, siendo importante rastrear las solicitudes o peticiones para construir el flujo de la ejecución del código [37].
- **Comprobación de salud:** Es importante saber frecuentemente el estado de un nodo para poder intervenir de forma rápida en caso de existir un problema debido a que monitorear la infraestructura de producción es uno de los puntos clave dentro de los equipos de operación. Describen las reglas pertinentes para poder determinar en qué estado se encuentra un punto de codificación. Por otro lado, podrían ser utilizados para decretar si un error de ser eliminado y sustituido por otro elemento por medio de mecanismos automatizados, permitiendo reducir el tiempo que estuvo inactivo el servicio [37].

En la actualidad Eclipse Microprofile consta de cuatro versiones cada versión con su respectiva enumeración conformadas por los diferentes marcos de ejecución como Open Liberty, Thorntail, Payara, KumuluzEE, WildFly Swam, TomEE y Helidon asi como también su adecuada versión de java 8 y 11 para poder ser desarrollados cada uno descritos como se pueden observar en la Tabla 7, Tabla 8, Tabla 9 y Tabla 10 [38]:

Tabla 7 Descripción Microprofile versión 1

MicroProfile Version 1	Runtime Micropfrofile	Java Versión
1.2	Open Liberty	8 – 11
	Thorntail V2	8
	Payara Micro	8
	KumuluzEE	8
	WildFly Swarm	8
	Apache TomEE 8.0.0-M3	8
	Helidon	8
1.3	Open Liberty	8 – 11
	Thorntail V2	8

	Payara Micro	8
	KumuluzEE	8
	Apache TomEE 8.0.0-M3	8
1.4	Open Liberty	8 – 11
	Payara Micro	8
	KumuluzEE	8
	Apache TomEE 8.0.0-M3	8

Tabla 8 Descripción Microprofile versión 2

MicroProfile Version 2	Runtime Micropfrofile	Java Versión
2.0	Open Liberty	8 – 11
	Payara Micro	8
	KumuluzEE	8
	Apache TomEE 8.0.0-M3	8
2.1	Open Liberty	8 – 11
	Thorntail V2	8
	Payara Micro	8
	KumuluzEE	8
	Apache TomEE 8.0.0-M3	8
2.2	Helidon	8
	Thorntail V2	8 – 11
	Payara Micro	8
	KumuluzEE	8
	Open Liberty	8 – 11

Tabla 9 Descripción Microprofile versión 3

MicroProfile Version 3	Runtime Micropfrofile	Java Versión
3.0	Open Liberty	8 – 11
	Thorntail V2	8 – 11
	KumuluzEE	8
	Helidon	8
3.2	Open Liberty	8 – 11
	Thorntail V2	8 – 11
	Payara Micro	8 – 11
	KumuluzEE	8
	WildFly	8 – 11
	Quarkus	8 – 11

	Helidon	11
3.3	Open Liberty	8 – 11
	Thorntail V2	8 – 11
	Payara Micro	8 – 11
	KumuluzEE	8
	WildFly	8 – 11
	Helidon	11

Tabla 10 Descripción Microprofile versión 4

MicroProfile Version 4	Runtime Micropfrofile	Java Versión
4.0	WildFly	8 – 11
	Open Liberty	8 – 11
	Payara Micro	8 – 11
4.1	Open Liberty	8 – 11
	WildFly	8 – 11

1.1.6.2. Heroku

Heroku es una plataforma digital perteneciente de la nube como una Plataforma como Servicio (PaaS) [39], se basa específicamente en contenedores virtuales dentro de un host compartido de ejecución fiable con todas las dependencias necesarias. Estos contenedores son llamados Dynos, lo que permite a los desarrolladores escalar sus aplicaciones web de forma momentánea solamente incrementando el número de contenedores o su tipo en el que se encuentra ejecutando la aplicación [40]. Posee una gran variedad de complementos que agiliza el desarrollo para las aplicaciones, además soporta diferentes lenguajes de programación como Java, Python, Node.js y PHP. Es un poderoso sistema que permite desplegar y poner en ejecución aplicaciones [41].

Las peticiones http realizadas remitidas a las aplicaciones desplegadas en Heroku son admitidas por un equilibrador de carga, el cual las remite a un conjunto de enrutadores denominado malla de enrutamiento, de aquí se forma la capa de enrutamiento de solicitudes http que tiene la función de acoger las peticiones y reenviarlas a los dynos de datos correspondientes como se representa en la Figura 20. Del mismo modo, se puede realizar una arquitectura simple alojada en Heroku con una infraestructura PaaS, aunque no sea una solución que permita el uso más eficiente de los recursos debido a que las peticiones se realizan de forma aleatoria hacía más de un contenedor al que debe recibir la solicitud http

[42]. Además, esta plataforma no permite desplegarse o navegar por su infraestructura por lo que brinda las herramientas necesarias para realizar una configuración adecuada para hacer uso de lo que se requiera en su utilización.

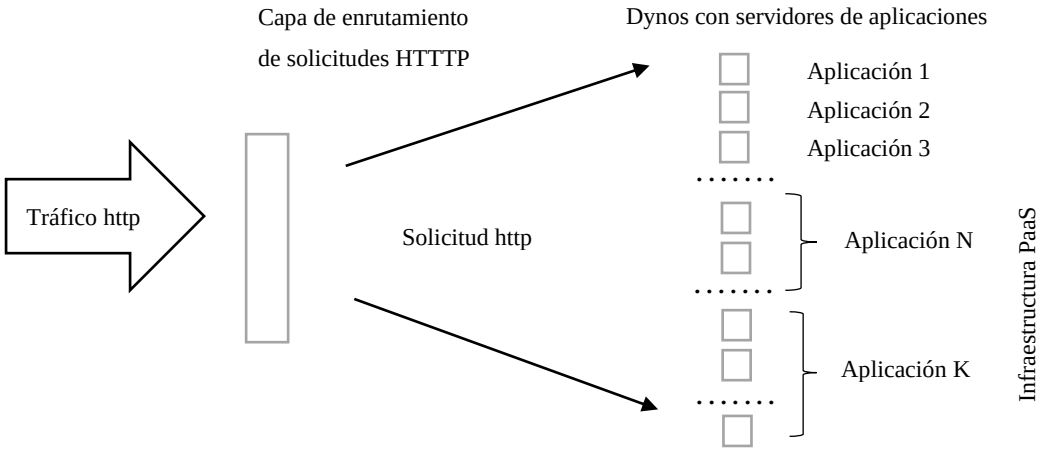


Figura 20. Arquitectura simplificada de Heroku PaaS

1.1.7. Lado del servidor y del cliente

Una aplicación está compuesta por dos lados que son el lado del servidor (back-end) y el lado del cliente (front-end). Sin embargo, una aplicación puede ser desarrollada mediante una pila completa es decir todo en una sola aplicación sin diferencia alguna de parte de ambas intervenciones.

El desarrollo en la parte del back end, hace énfasis en la parte del servidor de una aplicación, en la cual puede estar compuesta por tres partes muy importantes como: un servidor, una aplicación y una base de datos donde almacenar la información. Sin embargo, los usuarios no pueden observar la parte interna de la aplicación, pero aun así es la parte más esencial debido a que comunica las solicitudes de los usuarios a la información perteneciente a la base de datos. Por ejemplo, un back end puede ser desarrollado en Java, Php, Ruby, Python o .net [43].

Por otro lado, el desarrollo del front end se relaciona con la parte del cliente, en lo cual el objetivo principal es la visualización de la aplicación en un navegador o móvil por parte de los usuarios, por lo tanto, la única responsabilidad de los desarrolladores de front end es la apariencia o diseño del sitio web. De igual forma que el back end, el front end puede ser desarrollado por lenguajes como Html, Css, Javascript o JQuery [43].

1.2. Antecedentes

En el primer trabajo de investigación propuesto por González Ariel [20] titulado "Gestión de proyectos de vinculación mediante un aplicativo web con mensajería instantánea y arquitectura de servicios" tuvo como objetivo llevar un seguimiento sobre las actividades enfocadas en proyectos de vinculación que son indispensables y un requisito importante para obtener un título universitario. El proyecto de investigación trata sobre un aplicativo web desarrollado con arquitectura de microservicios, que sirve para optimizar y agilizar servicios de una aplicación utilizando el framework yii2 para la creación del frontend, mongoDB para el almacenamiento de datos y mensajería instantánea como telegram para la comunicación directa entre usuarios para el seguimiento de los proyectos de vinculación permitiendo conocer el estado de todos los proyectos y sus actividades en tiempo real y mantener las actividades de forma transparente y más comunicativa entre los participantes. El sistema desarrollado mediante las tecnologías mencionadas fue favorable, gratificante y fiable generando excelentes resultados al momento de monitorear y supervisar los proyectos de vinculación para aquellas personas que tenían relación directa o participación alguna en el proceso permitiendo así reducir el porcentaje de error administrativo al momento de manejar demasiada documentación física.

Un segundo estudio realizado por López José, en su trabajo de investigación de título "Arquitectura de Software basada en microservicios para desarrollo de aplicaciones web de la asamblea nacional" [25], tuvo como finalidad la revisión e implementación de una arquitectura basada en microservicios desarrollada con software libre como Spring Boot, Spring Cloud, Jenkins, Git, Docker para una aplicación web de la Asamblea Nacional. La arquitectura propuesta cumple con exactitud los requerimientos institucionales y tecnológicos deseados por la aplicación web de la Asamblea Nacional en comparación con la arquitectura monolítica con la que se desarrollaban antes las aplicaciones, además fue muy eficaz debido a que aprobó efectuar un desarrollo más enfocado en las funcionalidades del sistema siendo más ágil y rápido el proceso al momento de realizar peticiones de usuarios y visitantes en la aplicación. Además, se pudo lograr la separación de sus funcionalidades del sistema en pequeñas partes de manera que permiten ser manejables y en equipos independientes, cumpliendo con los criterios de evaluación y las tecnologías mencionadas.

En un tercer estudio titulado "Diseño de un prototipo de una arquitectura basada en microservicios para la integración de aplicaciones web altamente transaccionales. Caso: Entidades financieras" realizado por Diego Chicaiza [26], tiene como finalidad dar

lineamientos para el desarrollo de una arquitectura basada en microservicios para la integración de aplicaciones web orientadas a instituciones financieras altamente transaccionales y con una arquitectura muy compleja. Para el desarrollo de los microservicios se utilizó IBM como IDE, ESQL como lenguaje de programación para realizar las peticiones entre servicios para poder cumplir con la cantidad de transacciones y peticiones entre usuarios, aplicando componentes como front, middleware y backend para cumplir con los beneficios. A partir del desarrollo se obtuvieron buenos resultados con este tipo de arquitecturas ya que ofrecen flexibilidad, escalabilidad, estabilidad y velocidad dependiendo del contexto de uso en sistemas con bastantes requerimientos incluso utilizando diferentes tecnologías con relación al apartado anterior.

En un cuarto estudio comparativo entre arquitecturas titulado “Estudio comparativo entre una arquitectura con microservicios y contenedores dockers y una arquitectura tradicional (monolítica) con comprobación aplicativa” realizado por Pacheco José [44], tiene como objetivo realizar un análisis entre las ventajas y desventajas de cada arquitectura propuesta elaborando un prototipo de sistemas de módulos de gestión de productos y usuarios, calculando el rendimiento mediante Jmeter. Las tecnologías usadas para desarrollar un software monolítico fueron JEE como framework, Eclipse como software de desarrollo, Java, Jsp, HTML, JavaScript como lenguaje de programación, Jmeter como software para medir los recursos, apache-tomcat-8 (servidor) como software para servidor, careciendo de software para contenedor mientras que en un software de microservicios fueron Spring Boot como framework, eclipse como software de desarrollo, Java, Html, JavaScript como lenguaje de programación, Jmeter como software para medir los recursos, apache-tomcat-8 (imbebible) como software para servidor y Docker como software de contenedor. Los resultados de la comparación resultaron a favor de la arquitectura de microservicios obteniendo resultados muchos más favorables, rápidos y eficientes a diferencia de la arquitectura monolítica tanto en tiempo de peticiones de 0.5 segundos a 20 segundos, autenticación de usuario de 16 segundos a 278 segundos y cantidad de memoria utilizada de 180 Mb a 450 Mb respectivamente. Se puede concluir que al implementar la arquitectura de microservicios en un sistema mejora el tiempo de respuesta, aumenta la tolerancia a fallos y disminuye el tiempo de despliegue.

En un quinto estudio titulado “Propuesta de arquitectura de microservicios, metodología Scrum para una aplicación móvil de control académico: Caso Escuela Profesional de Obstetricia de la Universidad Nacional Mayor de San Marcos” [45], tiene como objetivo

principal exhibir un prototipo de un sistema móvil que permita perfeccionar la gestión y control académico mediante planificación y análisis relacionadas con la metodología Scrum y arquitectura de microservicios. Para llevar a cabo la investigación se utilizó la metodología scrum para elaborarlo detalladamente y para desarrollar el sistema se empleó react native para utilizar en dispositivos como Android e iOs, para el uso de microservicios se hizo uso de componentes funcionales como Spring Boot, OAuth2 para la seguridad de acceso a los servicios, Mysql para los datos y Rabbitmq para gestionar los eventos. La propuesta satisface los requerimientos de información e integración, además la arquitectura basada en microservicios genera independencia, acoplamiento, cohesión, flexibilidad, despliegue y descubrimiento de servicios por lo que se podría concluir que esta arquitectura es fiable y eficiente permitiendo que esta propuesta sea importante y reutilizable en otros sistemas que requieran mejorar el servicio.

En el sexto estudio analizado por Navarro Raúl y Cabrera Ricardo titulado “Aplicación basada en arquitectura de microservicios” [46], cuyo propósito es la implementación de una arquitectura de microservicios con una malla de servicios. Para el desarrollo de este proyecto se llevó a cabo el desarrollo de un sistema mediante la implementación de lenguajes que en la actualidad predominan como Python, Java, PHP, Spring Boot, bases de datos SQL y APIs REST. Además, fueron desplegados en Docker y Kubernetes. La aplicación tuvo una buena aceptación al tener una arquitectura de microservicios ya que pudo ser desplegada de forma más rápida sin tener inconveniente alguno, ocupa menos memoria y su implementación se dio de manera sencilla gracias a las tecnologías empleadas teniendo resultados muy favorables en comparación a que el sistema tenga una arquitectura monolítica.

En un séptimo estudio manifestado por Saransig Alexis titulado “Análisis de rendimiento entre una arquitectura monolítica y una arquitectura de microservicios – tecnología basada en contenedores” [8], tiene como enfoque principal realizar un análisis comparativo de rendimiento entre la arquitectura monolítica y la arquitectura de microservicios basada en contenedores. Para llevar a cabo la aplicación se utilizó tecnología de programación y desarrollo web actuales como Node.js y base de datos como Mysql y MongoDB para almacenar sus datos y Amazon Web Services para desplegar la aplicación. La aplicación fue puesto a prueba en diferentes escenarios de arquitecturas realizando solicitudes HTTP en donde las solicitudes procesadas por la arquitectura monolítica, dos solicitudes terminaron con error a diferencia de la arquitectura de microservicios en la cual todas terminaron con éxito, en relación al tiempo la aplicación de microservicios el tiempo se reduce en

comparación a la aplicación monolítica deduciendo que la eficiencia de los recursos ha mejorado, las solicitudes procesadas por segundos usando la aplicación de microservicios son mucho mejores y el tiempo de respuesta mejora en un 17% por lo que se puede dar mayor validez al experimento realizado en este proyecto. Llegando a la conclusión que una aplicación con arquitectura de microservicios desplegadas en contenedores y estas tecnologías mejora el rendimiento de los recursos en un sistema a diferencia de una aplicación con arquitectura monolítica que se procesa en una máquina virtual.

Finalmente, en un octavo estudio titulado “Increasing the dependability of IoT Middleware with Cloud Computing and Microservices” [47] tuvo como finalidad aplicar el patrón de arquitectura basada en microservicios en un middleware de IoT con servicios web en la arquitectura monolítica, analizando así las ventajas y desventajas que presente el enfoque y destacando la mejora en la disponibilidad, optimización y facilidad de mantenimientos que la arquitectura permita. Los módulos fueron desarrollados con PHP y Python, base de datos, servidores web y servicios de mensajería relaciones en un sistema operacional llamado RAISe el cual era un mecanismo que permitía simplificar objetos pequeños. Los autores obtuvieron resultados favorables teniendo una mejor latencia entre la comunicación de los módulos gracias a la implementación de la arquitectura de microservicios, un mejor resultado en la disponibilidad y optimización de los servicios web y un fácil mantenimiento de la aplicación desarrollada.

En relación con los estudios analizados se puede observar que todos utilizan tecnologías modernas, actualizadas y que predominan en el desarrollo de software, siendo diferentes lenguajes de programación para el desarrollo de microservicios, pero cumpliendo con los requerimientos propuestos a pesar de no existir un estudio directamente relacionado con la tecnología propuesta por el trabajo de investigación. Sin embargo, este tipo de arquitectura basada en microservicios como se pudo analizar puede ser empleado en aplicaciones web, aplicaciones móviles y aplicaciones IoT. Por último, varios estudios realizan una comparación entre los diferentes tipos de arquitecturas obteniendo resultados favorables enfocados en la arquitectura propuesta en este trabajo de investigación. Por lo que se podría manifestar que este tipo de arquitectura es recomendable para el diseño de una aplicación web de gran escala y transaccionalidad dentro de cualquier empresa u organización que lo requiera.

1.3. Bases legales

El trabajo de investigación se desarrolló teniendo en cuenta ciertas bases legales que permitieron regir la fiabilidad de la investigación, la cual está enfocada en el desarrollo de software, por lo que se estipulan las siguientes normativas o leyes: Código Orgánico integral penal [48], Decreto 1014 [49], Ley Orgánica de Educación Superior [50], Ley de la propiedad intelectual [51], Ley Orgánica de Emprendimiento e Innovación [52], Ley Orgánica de Telecomunicaciones [53]. Todas ellas corresponden a las leyes, decretos, códigos y normativas declaradas por el gobierno de Ecuador y son ejercidas dentro del territorio.

En el Código Orgánico Integral Penal en la sección tercera del tercer capítulo sobre los "Delitos contra los derechos del buen vivir" menciona los "Delitos contra la seguridad de los activos de los sistemas de información y comunicación" garantiza la seguridad de un software y sus componentes. Para esto, se describen varias penalizaciones en diferentes artículos como en el Art. 229 sobre la "Revelación ilegal de base de datos", en el Art 230 la "Intercepción ilegal de datos", en el Art 231 "Transferencia electrónica de activo patrimonial" y en el Art 232. sobre el "Ataque a la integridad de sistemas informáticos" [48]. Este Código Orgánico Penal mediante los artículos redactados permite brindar seguridad al proyecto de investigación que trata sobre el desarrollo de microservicios para la nube en caso de sufrir algún ataque, daño, intercepción o mal uso por medio de terceros que perjudiquen la integridad del software por medio de penalizaciones privativas de libertad.

Acerca de la ley Orgánica de Educación Superior, según el Art. 107 estipula que "La educación superior responda a las expectativas y necesidades de la sociedad, a la prospectiva de desarrollo científico" [50]. Esta ley nos indica que, es obligatorio que las instituciones educativas de nivel superior deben satisfacer las necesidades y expectativas que requiera la sociedad o estudiantes al momento de emplear o desarrollar algún proyecto investigativo que resulte beneficioso para ambas partes.

En la siguiente ley de Propiedad Intelectual, menciona en el Art. 4 "Se reconocen y garantizan los derechos de los autores y los derechos de los demás titulares sobre sus obras" y en el Art. 28 "Los programas de ordenador se consideran obras literarias y se protegen como tales." [51]. Esta ley permite evitar plagios o sustracción sobre algún software elaborado, ya que también es considerado como una obra que pertenece a un autor o dueño por lo que esta

ley favorecería a este trabajo de investigación evitar que terceros se hagan dueños de este, reconociendo como dueño o ser su titular.

Sobre la Ley Orgánica de Emprendimiento e Innovación que corresponde al segundo capítulo titulado “Políticas Públicas e Institucionales del Emprendimiento”, por medio del Art. 6 hace mención que “el Consejo Nacional para el Emprendimiento e Innovación – CONEIN es un organismo permanente y estratégico para promover y fomentar el emprendimiento, la innovación y la competitividad sistémica del país” [52]. Esta ley facilita promover e impulsar cualquier proyecto en general, además beneficiaría al proyecto de investigación ya que permitiría fomentar la comercialización de un software basado en arquitectura de microservicios que brinde mayor escalabilidad, rapidez y seguridad a sistemas de cualquier institución, empresa u organización que requiera de este servicio y cuyo desarrollo sea sencillo y sin mayor complejidad para los desarrolladores de software.

CAPÍTULO II

2. MATERIALES Y MÉTODOS

2.1.Delimitación de la investigación

El proyecto de investigación estuvo enfocado en un sistema de inventario para el negocio de fotografías “Fujifilm”. En donde se desarrolló una aplicación de inventario integrada con arquitectura de microservicios desplegada en la nube que permitió registrar, modificar y eliminar los productos que posee la empresa, logrando una automatización de tareas y mejorar el control. Al implementar esta arquitectura, se consideró como una innovación en un establecimiento comercial en la ciudad, puesto que se migró de un sistema tradicional a un software con una arquitectura actualizada y tecnologías actuales.

El trabajo tuvo una delimitación con relación a las herramientas o framework que se utilizaran para su desarrollo como lo son Eclipse Microprofile y Heroku. Al implementar estas herramientas permitieron desarrollar y desplegar microservicios que sean integrados a una aplicación web debido a que sus enfoques están relacionados en esta temática en específico.

2.2.Tipos de investigación

El trabajo de investigación fue de tipo documental ya que surgió la necesidad de extraer, estudiar y profundizar material teórico para la elaboración de la investigación, con apoyo de datos los mismos que fueron obtenidos de bases de datos bibliográficas, libros, tesis y otros materiales ilustrativos. Una vez aplicada esta técnica de investigación se obtuvo información para la elaboración referente al tema de arquitectura de microservicios basada en contenedores para la construcción del marco teórico y un posterior análisis a las características principales de sistemas de microservicios.

El estudio es de tipo descriptivo ya que “se selecciona una serie de cuestiones y se mide cada una de ellas independientemente, de forma tal de describir propiedades y características de lo que se investiga… Y puede ofrecer la posibilidad de llevar a cabo algún nivel de predicción.” [54].

Fue una investigación mixta debido a que se pretende utilizar y obtener datos cuantitativos y cualitativos. El enfoque cuantitativo “es aquel que utiliza de preferencia

información cuantitativa o cuantificable es decir medible para probar hipótesis con base en el análisis estadístico y numérico" [54].

En cambio, el enfoque cualitativo "es aquel que utiliza preferente o exclusivamente información de tipo cualitativo y cuyo análisis se dirige a lograr descripciones detalladas de los fenómenos estudiados" [54].

2.3.Métodos de la investigación

Para el trabajo de investigación se aplicó métodos de investigación como el método analítico – sintético el cual permitió analizar y sintetizar datos específicos acerca de las aplicaciones web que sean desarrolladas mediante la arquitectura de microservicios.

Método deductivo permitió empezar desde modelos o arquitecturas de software generales para luego pasar a las respectivas comparaciones que generen un análisis respectivo.

Método inductivo permitió realizar un enfoque orientado al análisis más específico sobre los resultados obtenidos, con el fin de ejercer una determinación sobre recursos, estrategias, herramientas que permitieron interferir en la demostración de dichos resultados referentes a la arquitectura de microservicios.

Método descriptivo permitió describir y especificar todas las variables con sus correspondientes indicadores, propiedades y características necesarias.

2.4.Variables de investigación

Para la ejecución del trabajo de investigación se tomaron en cuenta dos variables para el desarrollo y aplicaciones web las cuales se describirán a continuación.

Variable: Tecnologías para desarrollo de microservicios

Dentro de la variable de tecnologías para desarrollo de microservicios se tuvo presente las tecnologías empleadas, escalabilidad, comunicación y seguridad que se evaluó por medio de indicadores de tipo cuantitativos y cualitativos que brindaron resultados precisos referente a las cualidades de las tecnologías como se puede ilustrar la Tabla 11.

Tabla 11. Variable de tecnologías para desarrollo de microservicios

Variables	Definición conceptual	Dimensiones	Operacionalización	
			Indicadores	Tipo de Variable
Tecnologías para desarrollo de microservicios	Se analizan las diferentes tecnologías que se emplean para desarrollar sistemas basados en microservicios que sean de software libre.	Tecnologías empleadas	Rendimiento del software.	Cuantitativa
			Documentación	Cuantitativa
		Escalabilidad	Adaptabilidad del sistema.	Cualitativa
			Actualización del sistema	Cuantitativa
		Comunicación	Capacidad de respuesta	Cualitativa
			Tiempo de comunicación	Cualitativa
		Seguridad	Grado de seguridad del software en el funcionamiento del sistema	Cuantitativa
			Tiempo en la emisión de peticiones seguras	Cuantitativa

Dimensión: Tecnologías empleadas

A continuación, se describen indicadores como rendimiento del software y documentación que serán analizados dentro de las tecnologías empleadas.

- **Rendimiento del software:** Hace referencia al rendimiento que tendrá el software y el diseño de la aplicación basada en microservicios. Este indicador se mide en función del tiempo por milisegundos. Unidad de medida: milisegundos
- **Documentación:** Hace referencia a la documentación e información proporcionado por cada herramienta que permita el desarrollo de microservicios, facilitando diferentes tipos de librerías.

Dimensión: Escalabilidad

A continuación, se describen indicadores como adaptabilidad del sistema, actualización del sistema pertenencias a la escalabilidad.

- **Adaptabilidad del sistema:** Hace referencia a la capacidad de un software para adaptarse de forma eficiente y rápido para cambiar las circunstancias ante cualquier cambio. Porcentaje de efectividad.

- **Actualización del sistema:** Hace referencia a la facilidad que una aplicación puede actualizarse de forma eficaz sin ninguna complicación mediante su proceso. Porcentaje de efectividad.

Dimensión: Comunicación

A continuación, se describen indicadores como capacidad de respuesta, tiempo de comunicación que forman parte esencial de la comunicación.

- **Capacidad de respuesta:** Hace referencia a la capacidad de respuesta entre los servicios consumidos del software al momento de generar peticiones. Unidad de tiempo: minutos
- **Tiempo de comunicación:** Hace referencia al mejor tiempo de comunicación entre servicios ubicados en diferentes contenedores para tener un mejor rendimiento. Unidad de tiempo: minutos

Dimensión: Seguridad

A continuación, se describen indicadores como grado de seguridad del software en el funcionamiento del sistema, tiempo en la emisión de peticiones seguras pertenecientes a la seguridad.

- **Grado de seguridad del software en el funcionamiento del sistema:** Hace referencia al grado de seguridad que un software presenta en su funcionamiento acorde a los procesos que se ejecutan. Unidad de grado porcentual
- **Tiempo en la emisión de peticiones seguras:** Hace referencia al mejor tiempo de comunicación entre servicios ubicados en diferentes contenedores para tener un mejor rendimiento. Unidad de tiempo: minutos

Variable: Aplicaciones web

Dentro de la variable de aplicaciones web se tuvo presente el rendimiento y la calidad del proceso que evaluó el rendimiento y la calidad que tuvo la aplicación una vez desplegada y desarrollada con la arquitectura de microservicios mediante indicadores que ayudaron a obtener datos claros como se visualiza la Tabla 12.

Tabla 12. Variable de aplicaciones web

Variables	**Definición conceptual**	**Dimensiones**	**Operacionalización**	
			Indicadores	**Tipo de Variable**
Aplicaciones web	Se analiza lo referente a procesos, rendimiento y calidad que posee una aplicación web	Rendimiento del proceso	Tiempo de retrabajo.	Cuantitativa
			Defectos hallados en la verificación del software.	Cualitativa
			Tiempo no productivo.	Cuantitativa
			Frecuencia de despliegue.	Cuantitativa
		Calidad del proceso	Tiempo medio entre fallos.	Cuantitativa
			Defectos encontrados en el entorno	Cualitativa

Dimensión: Rendimiento del proceso

Se utilizaron indicadores como el tiempo de retrabajo, defectos hallados en la verificación del software, tiempo no productivo y frecuencia de despliegue que forman parte del rendimiento del proceso.

- **Tiempo de retrabajo:** Hace referencia al rendimiento que tendrá el proceso en ejecución ya desplegado de la aplicación basada en microservicios. Este indicador se mide en función del tiempo por segundos o minutos. Unidad de tiempo: segundos
- **Defectos hallados en la verificación del software:** Hace referencia a los defectos encontrados luego que el software haya sido verificado. Grado de errores porcentuales
- **Tiempo no productivo:** Es el tiempo en el que la aplicación está en modo inactivo dentro de la nube sin recibir ninguna petición o solicitud. Unidad de tiempo: segundos
- **Frecuencia de despliegue:** Hace referencia al rendimiento que tendrá el proceso en ejecución ya desplegado de la aplicación basada en microservicios. Este indicador se mide en función del tiempo por segundos o minutos. Unidad de medida: kb/s

Dimensión: Calidad del proceso

Se describieron indicadores como el tiempo medio entre fallos y defectos encontrados en el entorno las cuales forman parte de la calidad del proceso

- **Tiempo medio entre fallos:** Hace referencia al promedio de tiempo que transcurre entre una falla y la que sigue, es decir el tiempo en que algo que funciona hasta que falla y surge la necesidad de ser reparado. Unidad de tiempo: segundos
- **Defectos encontrados en el entorno:** Hace referencia a los defectos que se puedan encontrar mediante la arquitectura de microservicios en el entorno de la aplicación, con sus procesos de forma independiente. Si/No

2.5.Técnicas e Instrumentos de recolección de datos

Entrevista: Se aplicó al jefe de la empresa Fujifilm con la finalidad de obtener cada proceso detalladamente, inconvenientes que se presentan en el control de inventario. De tal forma se obtuvieron requerimientos funcionales y no funcionales para el desarrollo de los microservicios.

Especificación de requerimientos de software IEEE 830: Este formato aportó una descripción detallada del comportamiento de cada microservicio que se desarrolló y la interacción entre usuarios.

2.6.Técnicas de procesamiento y análisis de datos

Los datos obtenidos y utilizados para el desarrollo de la investigación fueron obtenidos mediante la toma de los requerimientos funcionales y no funcionales a través de la especificación de requerimientos de software IEEE 830, resultado que fue obtenido por medio de la entrevista o encuesta que se le realizo al jefe de la empresa con la finalidad de comprender cada proceso, inconveniente, herramienta con la que laboran dentro de la empresa.

2.7.Normas éticas

Para el presente trabajo de investigación se acataron todas las normativas establecidas por la Pontificia Universidad Católica del Ecuador Sede Esmeraldas, con el propósito de satisfacer con las normas de grado que decreta la institución. De igual forma, se cumple con el derecho de autor de cada trabajo o fuente de información valiosa la cual sirvió para aclarar y enriquecer definiciones referentes al tema planteado para el proyecto, por lo que se procedió

a colocar en cada párrafo o idea las citas correspondientes acorde a los trabajos obtenidos por fuentes bibliográficas y repositorios como Scopus, ACM, Xplorer IEEE enfocados en el desarrollo de microservicios para la nube mediante Eclipse Microprofile y Heroku.

Por otro lado, el trabajo de investigación emerge ante la poca información que existe sobre la arquitectura de microservicios integradas en las aplicaciones web nativas en la nube mediante herramientas específicas las cuales permitirán el desarrollo del proyecto. Además, se tendrá en cuenta los principios de ética profesional para la elaboración del proyecto en el que no afecte la investigación durante el desarrollo.

CAPÍTULO III

3. RESULTADOS

En el presente apartado se exponen los resultados que se obtuvieron durante la ejecución del proyecto de investigación. Para llevar a cabo el desarrollo de la investigación fue imprescindible realizar una entrevista, referenciado al formato de modelo estándar IEEE STD 830, siendo un objetivo necesario para obtener y comprender las funciones y procesos, a partir de este punto se obtuvieron resultados para los requerimiento funcionales y no funcionales con los que pueda contar la investigación.

Para el desarrollo de los microservicios se utilizó el IDE propio de Eclipse, herramienta apropiada para el uso del framework de Eclipse Microprofile ya que brinda facilidades para la creación de microservicios a partir de una plantilla base y la plataforma antes definida de Heroku**,** para conocer la compatibilidad y factibilidad que presentan estas dos tecnologías y se puede observar en el **repositorio público de GitHub**.

3.1.Requerimientos

Todos los requerimientos fueron obtenidos por medio de la entrevista y el formato de especificación especializada para requerimientos IEEE830, y se pueden observar en el anexo 2 y 3.

- **Definiciones, acrónimos y abreviaturas**

Nombre	**Descripción**
Usuarios	Persona que se encargará de la gestión y monitoreo de los procesos.
RNF	Requerimiento no funcional
RF	Requerimiento funcional

- **Descripción general**
    - **Perspectiva del microservicio:** Cada microservicio será desarrollado mediante dos herramientas como “Eclipse Microprofile” y “Heroku” para tener un diseño y un desarrollo más ligero a partir de sus métricas.

- **Características**

Tipo	**Desarrollador**
Usuarios	Asignación de administrador, asignación de empleados, actualización de usuarios.

Tipo	**Administrador**
Usuarios	Creación de productos, monitoreo de inventario.

Tipo	Empleados
Usuarios	Inspección de productos en stock.

- **Restricciones**
    - Los microservicios deben tener una implementación y programación ligera.
    - Deben tener la capacidad de responder peticiones frecuentemente.

3.2.Diseño de microservicios

Este proyecto tuvo como enfoque principal el desarrollo microservicios mediante Eclipse Microprofile y fue desplegado en la plataforma de Heroku. Cada microservicio tiene funcionalidades diferentes, siendo utilizados para agregar, actualizar, eliminar y administrar productos o usuarios, por lo que los usuarios pueden solicitar y revisar el stock registrado hasta el momento.

Un sistema de inventario es una de las herramientas más importante en cualquier empresa que permite llevar un control más estricto y de forma actualizada, así como también una facturación de la empresa con un porcentaje de error mínimo para el control total de toda la empresa. Además, ayuda a llevar información más específica de todo lo que contenga la empresa sobre sus productos, costos y categorías. Este proyecto puede ser utilizado por cualquier organización para la administración de su negocio.

Un microservicio contiene tres tipos de roles, administrador, supervisor o jefe y empleado.

- El administrador será el responsable de agregar el rol para alguna persona nueva.
- El supervisor puede llevar un control de todos los movimientos que se han hecho en el sistema, una auditoria referente a sus productos ingresados, actualizados o eliminados, así como también actualizaciones de costos.
- El empleado puede realizar una visualización de productos en stock.

La aplicación tiene las siguientes características principales:

- Gestión de productos: el administrador podrá registrar y enviar un producto a inventario semanalmente, actualizar costo o cantidad del producto.
- Registro de usuario: el administrar puede registrar a un empleado como usuario de venta, asignará un nombre de usuario que tendrá que ser el mismo como la dirección del correo electrónico, establecer una contraseña y asignar el rol al nuevo usuario.

Todo usuario recién creado podrá iniciar sesión en el sistema con las credenciales establecidas.

- Revisión y aprobación de inventario: el supervisor puede revisar todo el inventario y productos ingresados.

3.3.Arquitectura de microservicios

El sistema implementa una arquitectura de microservicios que es una de las arquitecturas más actuales y con gran impacto en las empresas para el desarrollo y despliegue de aplicaciones web. A continuación, la Figura 21 muestra la arquitectura lógica del sistema de inventario.

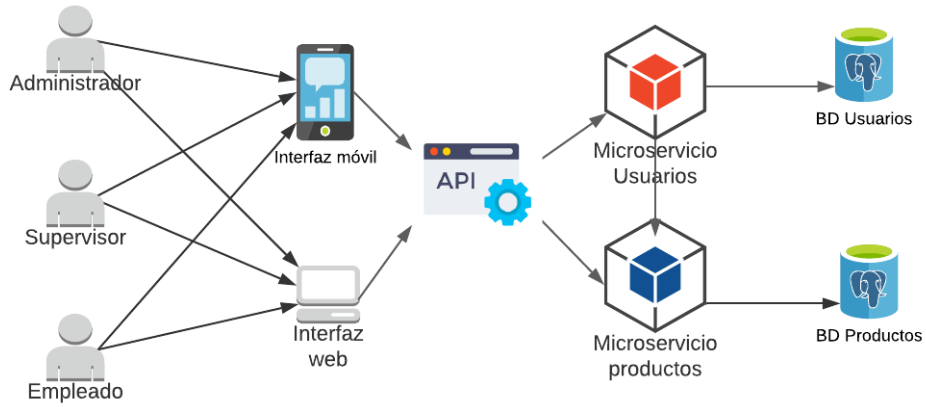


Figura 21. Diagrama de los microservicios y el sistema

El uso de esta arquitectura ayuda a estructura de forma lógica e independiente cada servicio de la aplicación con sus responsabilidades y requerimientos establecidos, dado que su objetivo principal es desarrollar la aplicación en conjunto de servicios con operaciones definidas y autónomas.

3.4.Diagrama de caso

Es imprescindible el desarrollo del diagrama de caso referentes al análisis estructurado para así conseguir un mejor punto de vista del proyecto.

Un caso de uso proyecta una interacción entre la aplicación y la intervención de los usuarios. En este proyecto existen tres tipos de usuarios: administrador, supervisor y empleado:

- El administrador es encargo de iniciar sesión, registrar usuario y cerrar sesión.
- El supervisor es responsable de iniciar sesión, controlar los procesos del sistema, agregar, actualizar o editar, eliminar producto y cerrar sesión.

- El empleado es responsable de iniciar sesión, visualizar los productos y cerrar sesión

La Figura 22 muestra el diagrama de casos de uso del sistema de inventario.

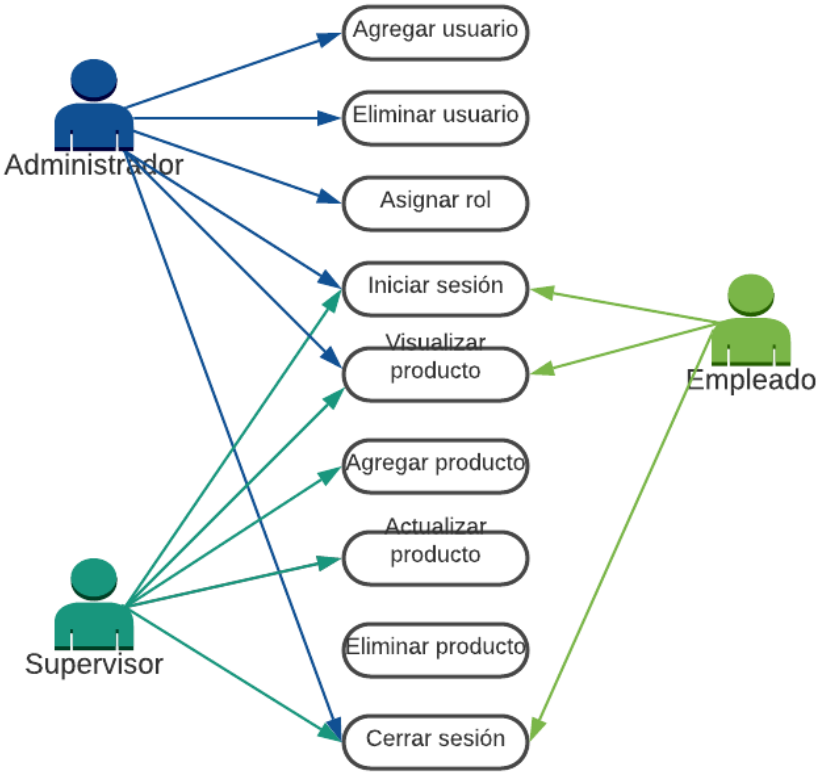
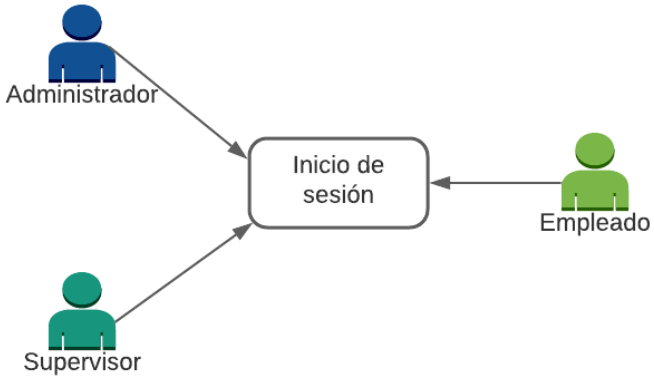


Figura 22. Diagrama de caso de uso del sistema

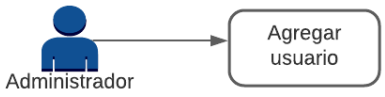
Cada caso de uso tiene una respectiva descripción y es presentada a continuación por cada integrante generado.

Caso de uso de inicio de sesión

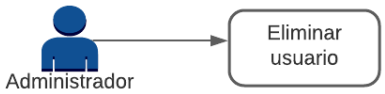


El caso de uso de inicio de sesión permite al administrador, supervisor, usuario iniciar sesión en el sistema. El sistema valida el nombre de usuario y contraseña ingresados por el usuario permitiendo ver la pantalla explicita al administrador, supervisor o empleado según su rol.

Caso de uso de agregar y eliminar un usuario:

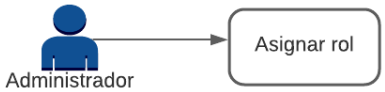


El sistema permite al administrador registrar un nuevo usuario.



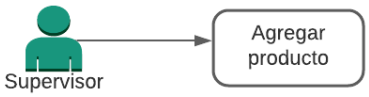
El sistema permite al administrador eliminar un nuevo usuario.

Caso de uso de asignar un rol a usuario:

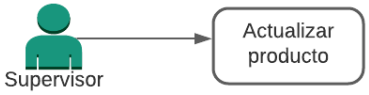


El sistema permite al administrador asignarle un rol a un usuario.

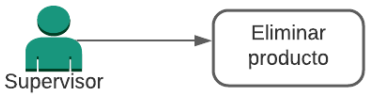
Caso de uso de agregar, actualizar, eliminar y visualizar un producto:



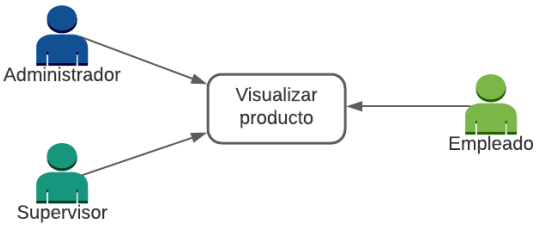
El sistema permite al supervisor agregar un producto.



El sistema permite al supervisor actualizar un producto.

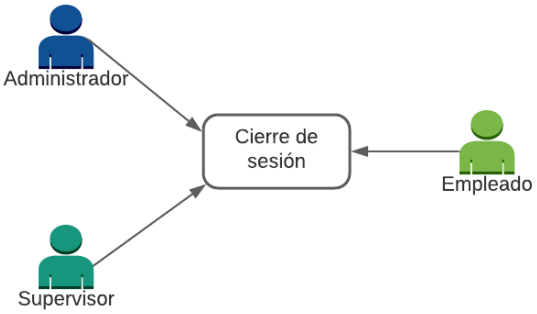


El sistema permite al supervisor eliminar un producto.



El sistema permite al administrador, supervisor y empleador visualizar un producto.

Caso de uso de cierre de sesión:



El sistema permite a todos los usuarios cerrar sesión.

3.5.Diagramas de secuencia

El diagrama de secuencia para la realización de inicio de sesión para el proyecto.

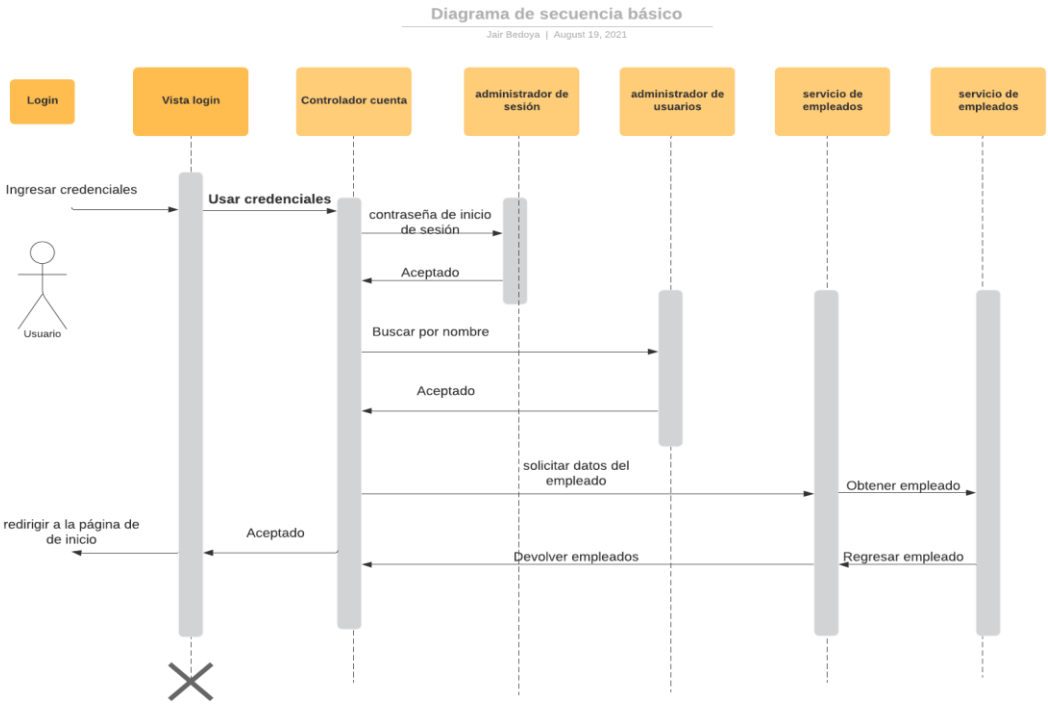


Figura 23. Diagrama de secuencia para la realización de inicio de sesión para el proyecto

El diagrama de secuencia para el desarrollo de registro de empleados.

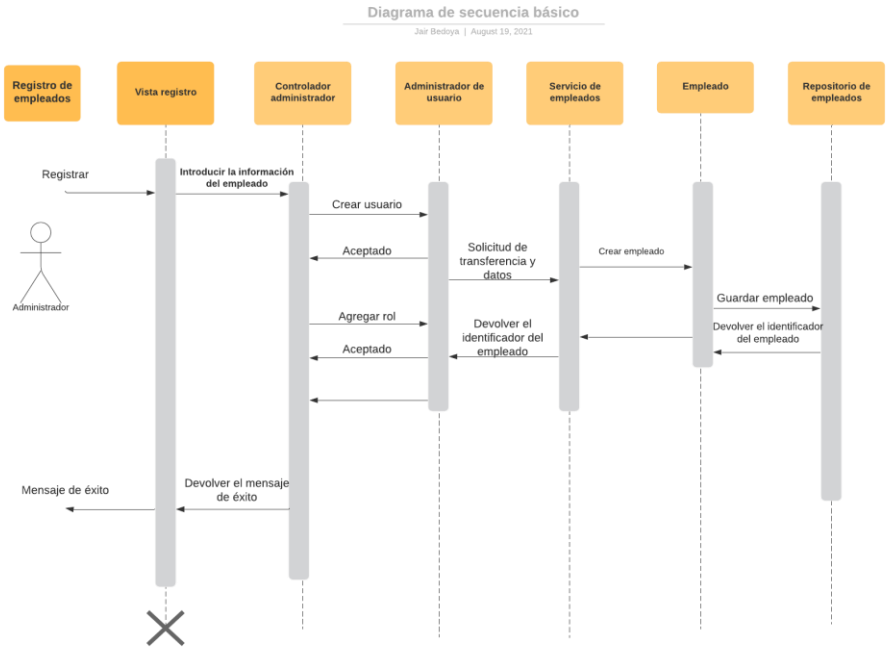


Figura 24. Diagrama de secuencia para el desarrollo de registro de empleados

El diagrama de secuencia para agregar, actualizar y eliminar productos.

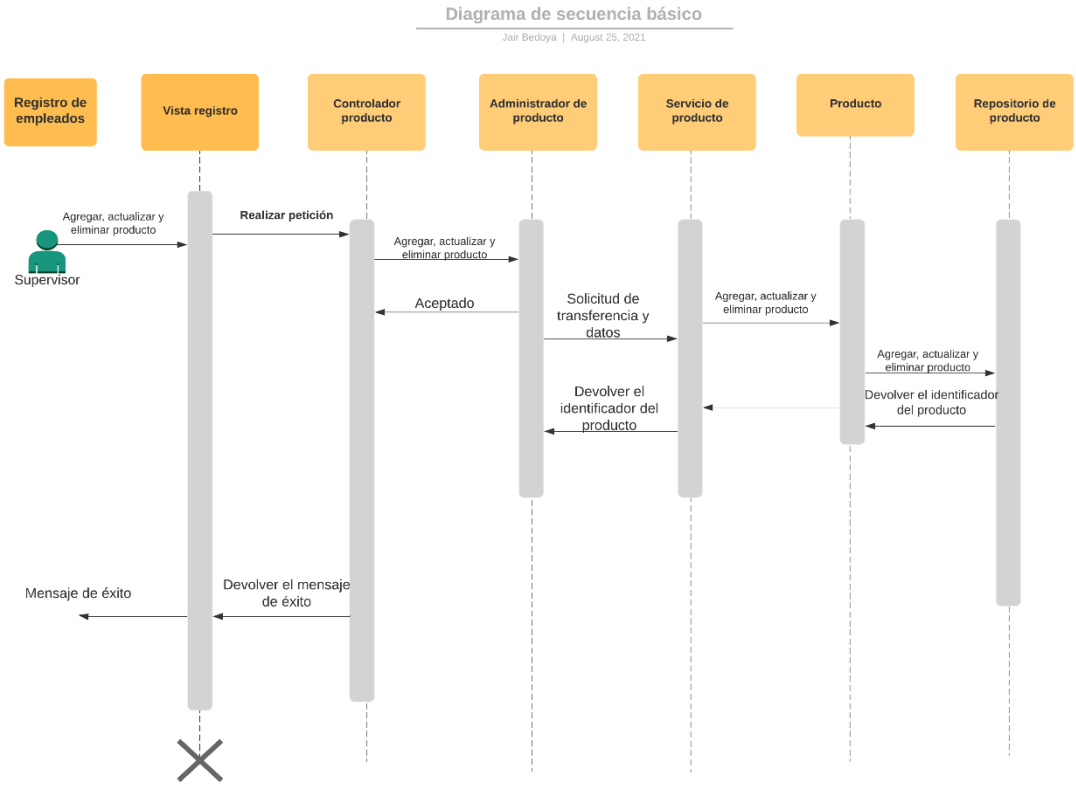


Figura 25. Diagrama de secuencia - Crud

3.6.Base de datos

El proyecto fue diseñado con una base de datos relacional PostgresDB dentro de la plataforma Heroku, permite la creación dentro del proyecto y su comunicación es directa entre la fuente de datos y el microservicio mediante el repositorio y las variables de entorno agregadas. A continuación, se muestra el diagrama de la base de datos:

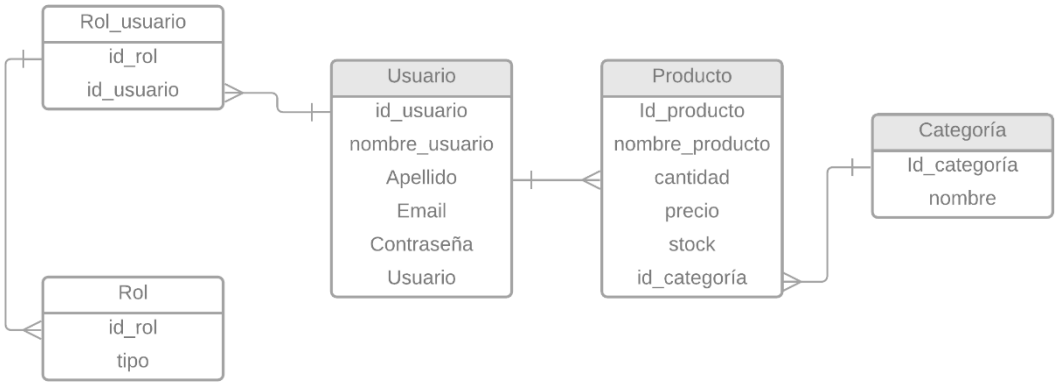


Figura 26. Diagrama relacional de Base de Datos Postgres

3.7.Selección del framework de microservicio

Debido a que la aplicación constaba de varios microservicios que se empaquetaran en contenedores Docker y se ejecutaran en un entorno de orquestación de contenedores dentro de Kubernetes se debieron abordar ciertas características especiales dentro de ellas por lo que se escogió de manera correcta un framework que cumpla con estas características.

Estos puntos claves se enfocan en el tiempo de inicio y el tamaño de memoria o del Jar que se genera al momento de utilizar un framework, sin embargo, el objetivo principal fue seleccionar uno que genere archivos tan pequeños como sea posible, empaquetando todo en un único archivo llamado JAR. Por otro lado, fue de suma importación observar cuanta memoria utilizaba un microservicio mientras se daba la ejecución por lo que menos memoria consuma un microservicio era mucho mejor y así tener una arquitectura de microservicios rentable.

Para seleccionar que framework escoger, se utilizaron microservicios de la página oficial de eclipse de una aplicación de muestra para poder realizar una comparación minuciosa enfocado en esas dos características especiales como lo son el tamaño del JAR y el tiempo de inicio para los siguientes framework: Liberty, WildFly, Payara, TomEE y KumuluzEE, y estos archivos de prueba se encuentran alojados en el **repositorio público de Github-runtimeTesis**.

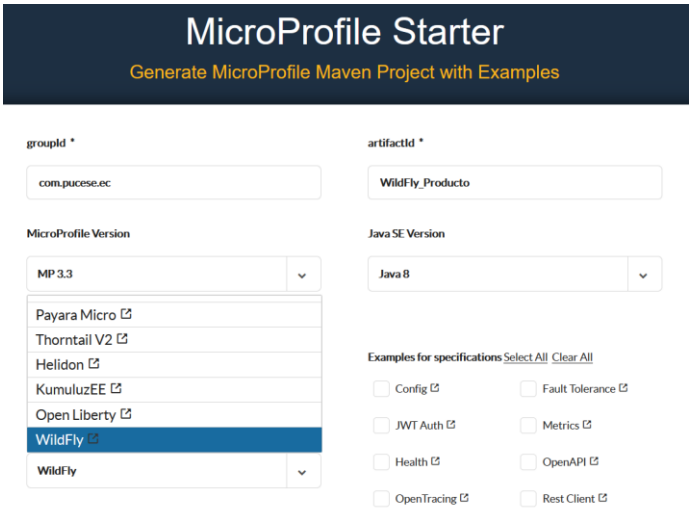


Figura 27. Generador de microservicios

Los resultados que se obtuvieron y analizaron para la selección del framework se muestran en la siguiente tabla:

Tabla 13. Comparación de Framework Microprofile

Framework	Tamaño Jar/Mb	Tiempo de inicio/s
Liberty	35	7
WildFly	65	6
Payara	33	5
TomEE	35	3
KumuluzEE	11	2

A partir de los resultados se puede observar que hay diferencias bastantes considerables entre los diferentes framework que se pueden utilizar en Microprofile, estos datos han sido repetidos por cuenta propio y se obtuvieron resultados muy parecidos. Tener en cuenta que el tiempo de inicio, el tamaño de la memoria y el consumo de memoria son características muy importantes que se deben tener en cuenta una vez que se empiece a desarrollar una arquitectura de microservicios en la nube. Por lo tanto, para esta investigación solo se tuvo en cuenta el framework KumuluzEE por su rapidez, poco espacio y poco consumo de memoria.

3.8.Desarrollo de microservicios

Luego de tener el framework adecuado para el desarrollo de los microservicios se utilizó la herramienta de programación Eclipse siendo una plataforma de software que contiene un conjunto de librerías para el desarrollo de aplicaciones web, así como librerías de Java EE que permiten la creación de microservicios. Se trabajó con el framework Eclipse Microprofile para el desarrollo de microservicios debido a la facilidad que brinda para la creación a partir de una plantilla inicial con todas sus dependencias requeridas.

En primer lugar, para el desarrollo de cada microservicio mediante un análisis de requerimientos funcionales y no funcionales, lo que permitió estudiar los procesos de administración de inventario realizados por la empresa Fujifilm. Partiendo de este análisis y los procesos no sistemáticos implicados en el inventario, se obtuvieron 3 entidades principales (usuario, productos y categoría), las cuales poseen información del usuario con su rol específico e información del producto dependiendo de su categoría.

Cada microservicio fue creado a partir de un proyecto Maven en donde se incluyeron algunas dependencias como una interfaz API que permitía la comunicación con otros servicios, una persistencia en la base de datos y caché, posteriormente se procedió a la creación de las clases de entidades, repositorios, controladores, recursos con sus respectivos parámetros para cumplir con los procesos de la administración de inventario del sistema web

En el archivo pom.xml se encuentran las dependencias a utilizar para el desarrollo de cada microservicio del framework Eclipse Microprofile, jdk 8, base de datos postgres y por último hibernate para el mapeo objeto-relacional con sus respectivas versiones, tal y como se puede observar en la Figura 28.

```
<dependencies>

  <dependency>
    <groupId>org.eclipse.microprofile</groupId>
    <artifactId>microprofile</artifactId>
    <version>2.1</version>
    <type>pom</type>
    <scope>provided</scope>
  </dependency>

   <dependency>
  <groupId>javax</groupId>
  <artifactId>javaee-api</artifactId>
  <version>8.0</version>
  <scope>provided</scope>
  </dependency>

  <dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-agroal</artifactId>
  <version>5.4.3.Final</version>
  </dependency>

  <dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>42.2.5</version>
  </dependency>

</dependencies>
```

Figura 28. Dependencias para microservicios

Luego, lo que se procedió a realizar la configuración de la base de datos creando un archivo resources.xml dentro de la carpeta src/main/webapp/WEB-INF y se introdujo el siguiente código que son las variables y la configuración de la base de datos a utilizar como se puede observar en la Figura 29.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tomee>
    <Resource id="jdbc/my_blog_datasource" type="javax.sql.DataSource">
        JdbcDriver = org.postgresql.Driver
        JdbcUrl = jdbc:postgresql://localhost/postgres
        UserName = postgres
        Password = 123456
        jtaManaged = true
    </Resource>
</tomee>
```

Figura 29. Recursos de la base de datos

Una vez realizada la configuración a la base de datos se creó un archivo llamado persistence.xml dentro de la carpeta src/main/resources/META-INF y se introdujo el siguiente código como se visualiza en la Figura 30.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0"
             xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
    <persistence-unit name="myBlog_PU" transaction-type="JTA">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <jta-data-source>jdbc/my_blog_datasource</jta-data-source>


        <exclude-unlisted-classes>false</exclude-unlisted-classes>
        <properties>
            <property name="tomee.jpa.factory.lazy" value="true" />
            <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQL9Dialect" />

            <property name="hibernate.hbm2ddl.auto" value="update"/>

            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.format_sql" value="true" />
            <property name="hibernate.transaction.jta.platform" value="org.hibernate.service.jta.platform.internal.SunOneJtaPlatform"/>
        </properties>
    </persistence-unit>
</persistence>
```

Figura 30. Archivo persistence.xml

Archivo persistence.xml

Este archivo es la configuración estándar para JPA por lo que debe estar incluido en el META-INF del directorio debido a que define que proveedor es utilizado, el nombre de la unidad de persistencia y como deben ser asignadas las clases a las tablas de la base de datos.

Una vez terminada las configuraciones sobre las dependencias de los microservicios se procede a codificar el api con sus respectivos atributos como las entidades, repositorio y recursos.

Creación de la clase entidad

Dichas entidades pertenecen al tipo de la clase POJO (objeto java plano antiguo) ya que sirven para hacer énfasis el uso de clases sencillas sin la dependencia de algún framework en específico.

Dentro de la carpeta Entities creamos una nueva clase llamada Productos.java y agregamos el siguiente código, el cual posee la creación y el nombre de la entidad de la tabla en la base de datos, la consulta para obtener información desde la base de datos, variables o atributos de la clase producto y los campos que se van a crear en la base de datos luego de empaquetar el microservicio.

```java
package com.pucese.blog.entities;

import java.io.Serializable;
import javax.validation.constraints.Size;
import java.util.Objects;
import javax.persistence.*;


@Entity
@Table(name = "producto")
@NamedQueries({
        @NamedQuery(name = "Producto.findAll", query = "SELECT p FROM Producto p")
})
public class Producto implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
       @GeneratedValue(strategy = GenerationType.IDENTITY)
       @Column(name = "Id")
       private Long Id;

       @Column(name = "nombre", length = 50)
       private String nombre;

       @Column(name = "procantidadadduct_stock")
       private int cantidad;

       @Column(name = "disponible")
       private boolean disponible;

       @Column(name = "precio")
       private double precio;

       public Producto() {

       }

       public Producto(String nombre, int cantidad, double precio) {
           this.nombre = nombre;
           this.cantidad = cantidad;
           this.precio = precio;
           this.disponible = true;
       }
```

Figura 31. Clase entidad

- **Entity**.- Indica que la anotación es una entidad JPA.
- **Table**.- Se usó para nombrar la tabla en la base de datos.
- **NamedQueries**.- Se utilizó para agregar varias consultas.
- **NamedQuery**.- Se utilizó para definir la consulta por su nombre.
- **Id**.- Se utilizó para poder definir la clave principal.
- **GeneratedValue**.- Se utilizó para indicar que el Id debe generarse automáticamente.
- **Colum**.- Se usó para especificar la columna asignada en una propiedad persistente.

Creación de la lógica de negocio

Una vez codificada la clase de entidades, se procedió a crear un nuevo archivo llamada ProductoRepository dentro de la carpeta repositories y se agregó el siguiente código:

```
package com.pucese.blog.repositories;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import com.pucese.blog.entities.Producto;

import java.util.List;

@Stateless
public class ProductoRepository {

    @PersistenceContext(unitName = "myBlog_PU")
    EntityManager em;

    public List getAllProducto() {
        return em.createNamedQuery("Producto.findAll", Producto.class).getResultList();
    }

    public Producto findById(Long id) {
        return em.find(Producto.class, id);
    }

    public Producto create(Producto producto) {
        em.persist(producto);
        return producto;
    }

    public void update(Producto producto) {
        em.merge(producto);
    }

    public void delete(Producto producto) {
        if (!em.contains(producto)) {
            producto = em.merge(producto);
        }

        em.remove(producto);
    }
}
```

Figura 32. Repositorio de la clase producto

Esta clase importa o instancia los atributos de la clase “Producto” y se describen los métodos codificados a continuación:

- **PersistenceContext**.- Lo que hace es inyectar el EntityManager que se utilizará en tiempo de ejecución.
- **GetAllProducto**().- Este método recupera todas las publicaciones de la base de datos y devuelve la lista completa.
- **FindById**().- Este método encuentra un producto de la base de datos con ID y lo devuelve.
- **Update**().- Este método actualizará un producto existente en la base de datos.
- **Create**().- Este método creará un producto en la base de datos.
- **Delete**().- El método encontrará un producto en la base de datos y lo eliminará.

Creación del recurso

```java
package com.pucese.blog.resources;

import javax.enterprise.context.RequestScoped;
import javax.inject.Inject;
import javax.ws.rs.*;
import javax.ws.rs.core.*;
import com.pucese.blog.entities.Producto;
import com.pucese.blog.repositories.ProductoRepository;

@RequestScoped
@Path("producto")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)

public class ProductoResource {

    @Inject//conexion con la capa de datos
    ProductoRepository productoRepository;

    @GET//metodo para mostrar todos
    public Response getAllProducto() {
        return Response.ok().entity(productoRepository.getAllProducto()).build();
    }
    @GET
    @Path("{id}")
    public Response getPostById(@PathParam("id") Long id) {
        return Response.ok().entity(productoRepository.findById(id)).build();
    }
    @POST
    public Response create(Producto producto, @Context UriInfo uriInfo) {
        Producto productoId = productoRepository.create(producto);
        UriBuilder builder = uriInfo.getAbsolutePathBuilder();
        builder.path(Long.toString(productoId.getId()));
        return Response.created(builder.build()).build();
    }
    @PUT
    @Path("{id}")
    public Response update(@PathParam("id") Long id, Producto producto) {
        Producto updateProducto = productoRepository.findById(id);

        updateProducto.setNombre(producto.getNombre());
        updateProducto.setCantidad(producto.getCantidad());
        updateProducto.setPrecio(producto.getPrecio());
        updateProducto.setDisponible(producto.isDisponible());

         return Response.ok().entity(producto).build();
    }

    @DELETE
    @Path("{id}")
    public Response delete(@PathParam("id") Long id) {
        Producto producto = productoRepository.findById(id);
        productoRepository.delete(producto);
        return Response.noContent().build();
    }
}
```

Figura 33. Recurso de la clase ProductoRepositorio

- @**RequestScoped** La anotación indica que esta clase se creará una vez en cada solicitud.
- @**Path** la anotación identifica la ruta de URI a la que responde el recurso.
- @**Produces** La anotación convertirá automáticamente la respuesta al formato JSON.
- @**Consumes** La anotación convertirá automáticamente la cadena JSON publicada aquí en un Producto.
- @**Inject** La anotación inyecta el ProductoRepository..
- @**GET** La anotación asigna /posts la solicitud HTTP GET al getAll()método, que recuperará todas las publicaciones de la base de datos y devolverá la lista completa

Estructura del directorio del microservicio de producto

Una vez codificado y configurado el microservicio de productos, la estructura del directorio queda de la siguiente forma:

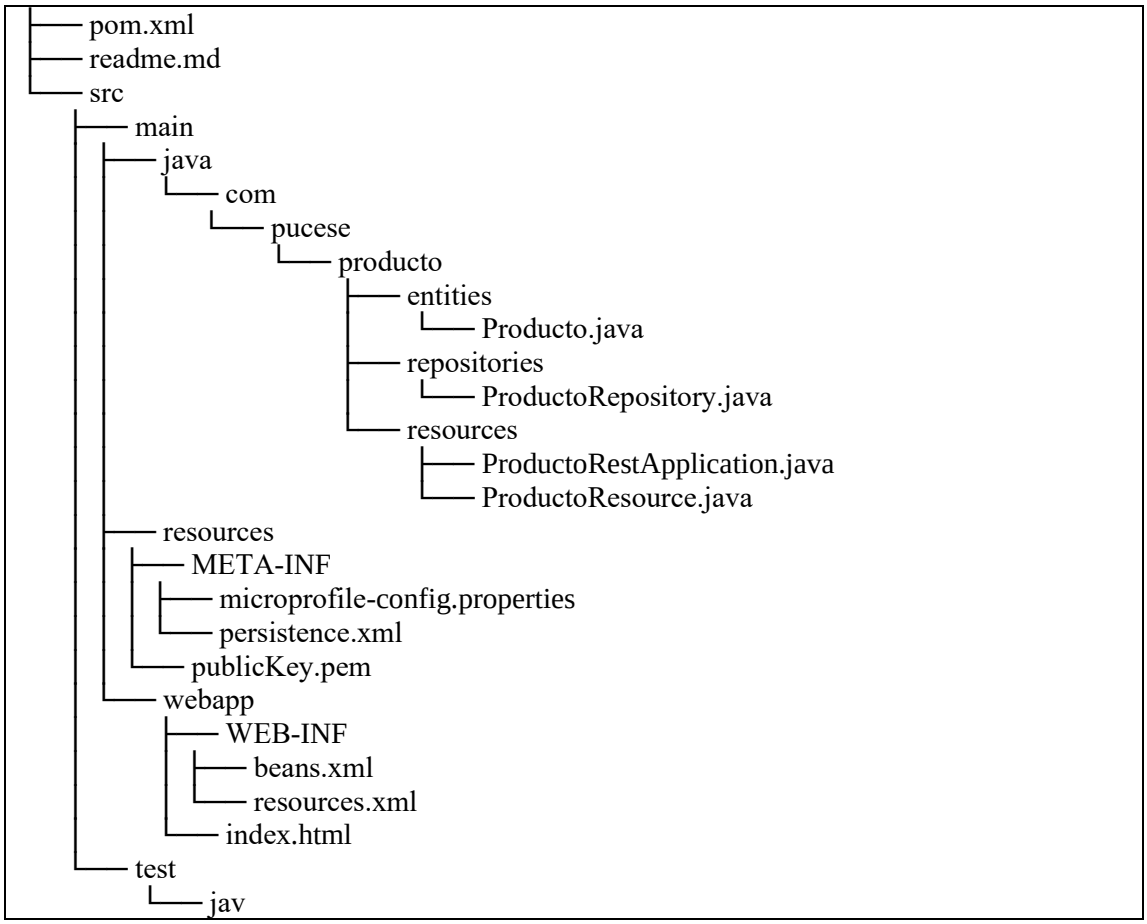


Figura 34. Directorio del proyecto

Seguridad del microservicio

Para implementar seguridad y poder realizar solicitudes HTTP, agregamos al archivo pom.xml las dependencias importantes de Cliente HTTP CXF

```
<dependency>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-rt-rs-client</artifactId>
    <version>3.1.13</version>
    <scope>provided</scope>
</dependency>
```

Una vez agregadas las solicitudes se realiza la codificación para poder obtener las peticiones de los clientes con sus respectivas credenciales

```
//Instantiate the client:
private final String ratingsUser = System.getProperty("api.access.user");
private final String ratingsPassword = System.getProperty("api.access.password");

private final WebClient webClient = WebClient.create("http://localhost:8080", ratingsUser, ratingsPassword, null);

    /**
    * Get the rating of a movie.
    *
    * @param id the user ID
    * @return an int between 0 (avoid) to 10 (masterpiece)
    */
private int getRating(final long id) {
   return webClient.reset()
          .path(RATING_PATH, id)
          .get(Integer.class);
}
```

Y dentro del IDE se puede observar de la siguiente forma:

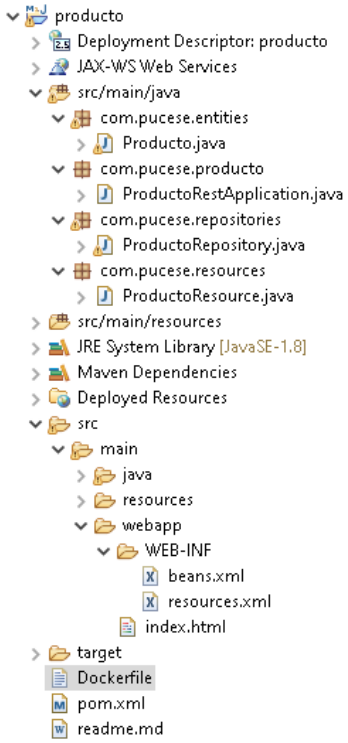


Figura 35. Estructura del microservicio

.

Prueba

Una vez construido nuestro microservicio de forma local se realizaron solicitudes para obtener información guardada en la base de datos y se obtuvo la siguiente información:

```
C:\Users\GamaPrinter\Desktop\proyecto_tesis\producto
λ curl -v http://localhost:8080/data/producto
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8080 (#0)
> GET /data/producto HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.63.0
> Accept: */*
>
< HTTP/1.1 200
< Date: Wed, 25 Aug 2021 08:16:24 GMT
< Content-Type: application/json
< Transfer-Encoding: chunked
< Server: Apache TomEE
<
[{"cantidad":100,"disponible":true,"id":1,"nombre":"Cámara fotográfica","precio":100.0}]* Connection #0 to host localhost left intact
```

Figura 36. Resultado prueba método Get

Prueba mediante la herramienta de Postman

Método Get

Para poder tener la información que se encuentra en la base de datos, se usa el método get como se puede observar en la codificado en la clase de ProductoResource a través del productoRepository e instanciado por la clase Producto con sus respectivos atributos.

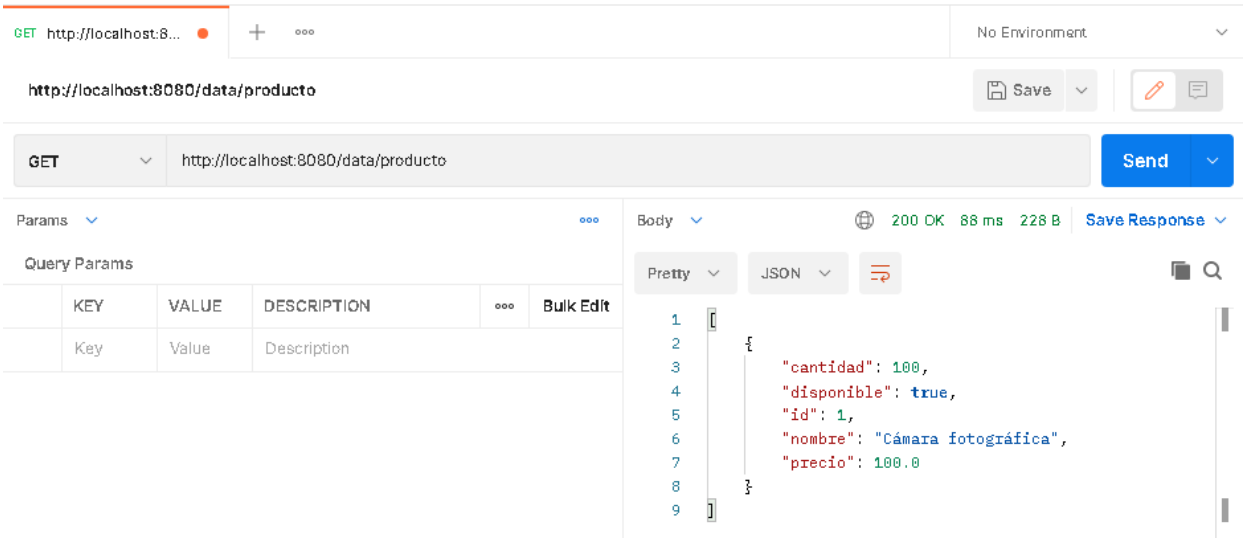


Figura 37. Petición Get

Método Post

Para comprobar que el crud del microservicio se encuentre funcionando de correcta manera, se hizo la prueba mediante el método post. Lo que se hizo fue colocar los mismos campos ya definidos en la base de datos y realizar la petición.

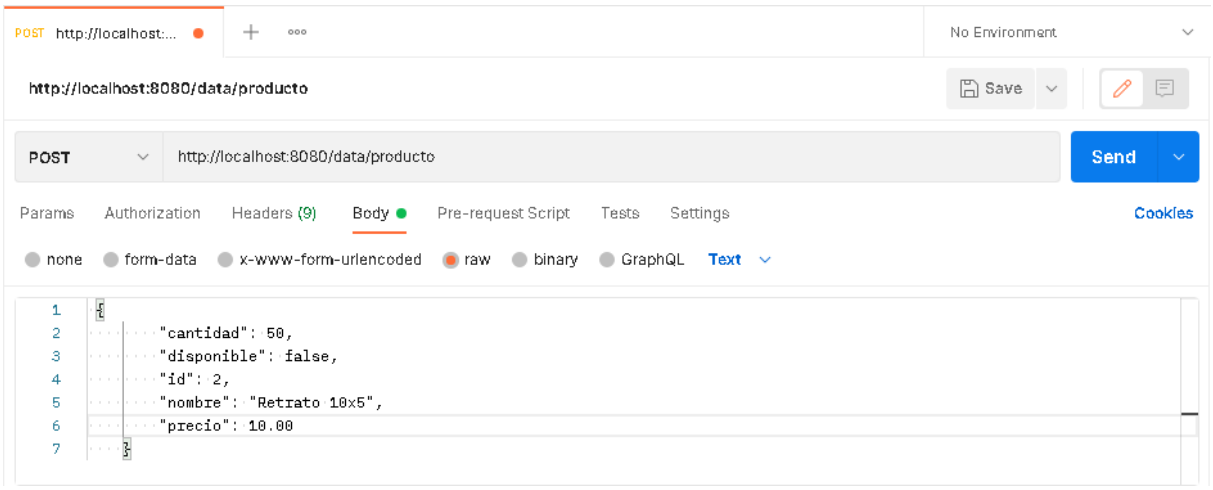


Figura 38. Petición Post

Y el resultado fue el siguiente como se puede observar en la Figura 39.

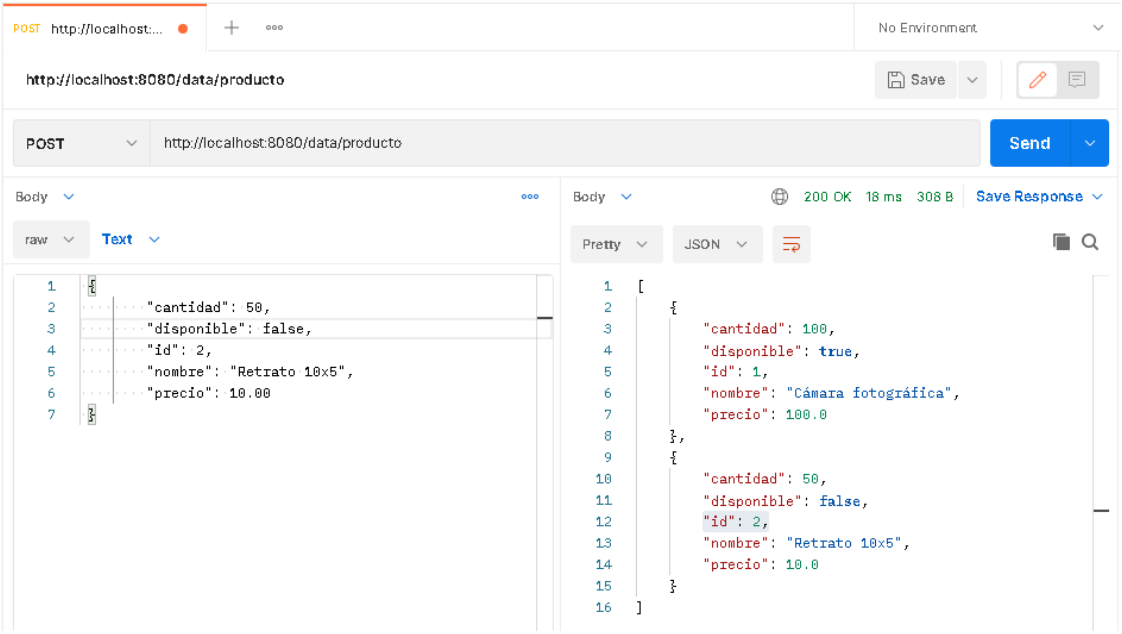


Figura 39. Petición Post resultado

Por último, hacemos una petición para poder eliminar un dato de la base de datos mediante el id del producto que se quiera eliminar como se puede observar en la Figura 40 y Figura 41, dando como resultado una petición satisfactoria.

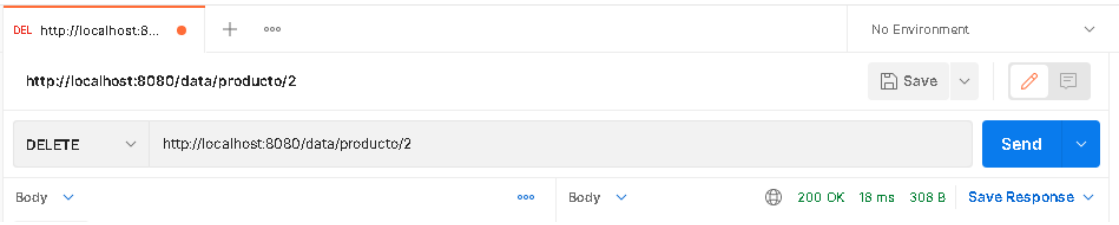


Figura 40. Petición Delete

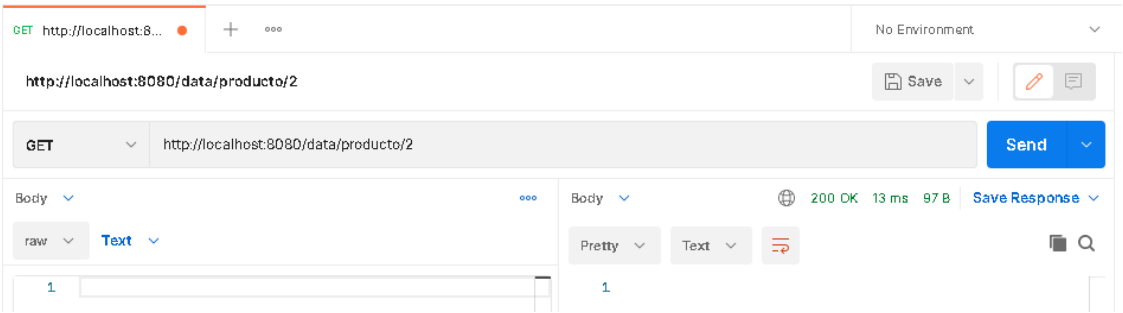


Figura 41. Petición Delete resultado

3.9.Despliegue de microservicio en la plataforma de Heroku

Luego de haber desarrollado los microservicios con todas sus dependencias definidas se procedió al despliegue de los microservicios (proyecto maven) en la plataforma de Heroku, esta plataforma cumple con múltiples características esenciales para el despliegue de una aplicación como facilidad de integración, portabilidad, poca complejidad y bajo costo. Para esto se necesitó tener una cuenta oficial en la plataforma y descargar el Heroku CLI que permite comunicarse directamente con el servidor de la nube.

Posteriormente de estar listo el microservicio nos dirigimos a la carpeta del proyecto que se desee desplegar desde la línea de comando, una vez adentro lo que se debe hacer es iniciar sesión con el comando “Heroku login” a lo que procederá a pedir credenciales de la cuenta propia. En caso de haber registrado ya la cuenta simplemente pedirá ingresar al navegador y loguearse como se puede observar en la Figura 42.

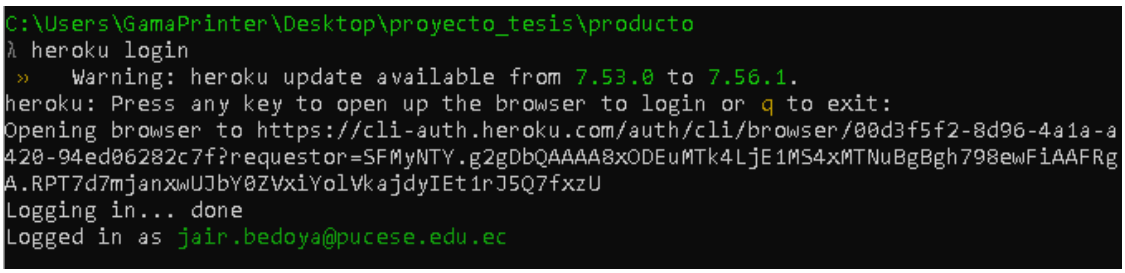


Figura 42. Iniciar sesión en Heroku

Heroku tiene implementado un método de doble autentificación para poder evitar cualquier conflicto entre cuentas y terceros involucrados por lo que para poder hacer uso de las herramientas de la plataforma es necesario darle acceso y permiso a todo como se puede observar en laFigura 43.

Posteriormente, se necesitó crear una aplicación para la plataforma de Heroku por lo que se procede a ingresar el comando “Heroku créate” o “Heroku create pucese-heroku” lo cual creara la aplicación con el nombre que se le asignó y será visible en la dirección web caso contrario como el primer comando se crea con un nombre aleatorio.

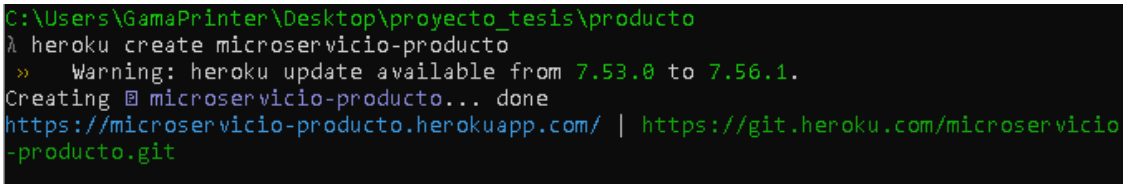


Figura 43. Comando para crear un proyecto en Heroku

Una vez terminado este paso se puede comprobar en la página oficial de Heroku (Dashboard) y ver la aplicación que se creó como se puede observar en la Figura 44.

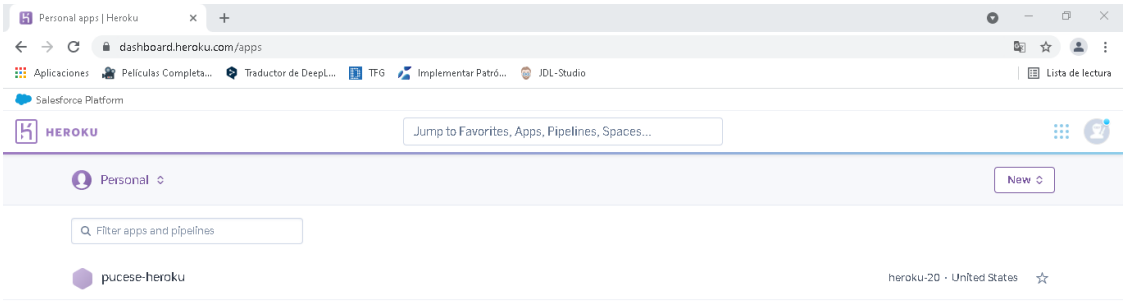


Figura 44. Dashboard Heroku

Con el comando "Heroku apps:info pucese-heroku" se puede tener información sobre la configuración definida en la plataforma, como la dirección web, el propietario, url del repositorio, los adicionales como base de datos y él microservicio será completamente levantado, tal como se puede observar en la Figura 45.

```
λ heroku apps:info pucese-heroku
 »   Warning: heroku update available from 7.53.0 to 7.56.1.
=== pucese-heroku
Addons:         heroku-postgresql:hobby-dev
Auto Cert Mgmt: false
Dynos:
Git URL:        https://git.heroku.com/pucese-heroku.git
Owner:          jair.bedoya@pucese.edu.ec
Region:         us
Repo Size:      0 B
Slug Size:      0 B
Stack:          heroku-20
Web URL:        https://pucese-heroku.herokuapp.com/
```

Figura 45. Información del proyecto en Heroku

Y se procedió a mandar el proyecto al repositorio de Heroku para poder visualizar los cambios en la aplicación, por lo que se puede notar se aceptaron todos los cambios y ninguno fue rechazado:

```
C:\Users\GamaPrinter\Desktop\proyecto_tesis\producto (master)
λ git push heroku master
Enumerating objects: 328, done.
Counting objects: 100% (328/328), done.
Delta compression using up to 4 threads
Compressing objects: 100% (305/305), done.
Writing objects: 100% (328/328), 169.15 MiB | 4.11 MiB/s, done.
Total 328 (delta 28), reused 0 (delta 0)
```

Figura 46. Publicación en repositorio local

CAPITULO IV

4. DISCUSION

Saransig Alexis [8] en su investigación realizó un análisis comparativo mediante un escenario de pruebas entre ambas arquitecturas y con las mismas tecnologías como Node.js, MySql, MongoDB y Amazon Web Services, en donde los resultados positivos se inclinaron más hacia la arquitectura de microservicios careciendo de errores en sus peticiones en un tiempo menor a diferencia de la arquitectura monolítica: Por esta manera se ha podido constatar por medio de una revisión bibliográfica que esta arquitectura de microservicios desplegada en contenedores y con tecnologías dan mejores resultados a una aplicación o sistema web en relación a las solicitudes procesadas y el tiempo de respuesta.

López José [25] en su trabajo de investigación desarrolló la aplicación con software libre, en donde esta arquitectura cumplió con perfección los requerimientos establecidos por la aplicación web, siendo más ágil y rápido el proceso al momento de solicitar peticiones debido a que sus servicios se encuentran completamente independientes. Esta información ha sido relacionada con el trabajo de investigación debido al desarrollo de los microservicios y su independización de los otros servicios sin tener interferencia alguna, siendo más factible y rápido sus procesos.

González Ariel [20] en su proyecto de tesis desarrolló su aplicación relacionada a proyectos de vinculación mediante tecnología y arquitectura de microservicios para optimizar y agilizar los procesos, las herramientas o framework que utilizo fueron yii2 para el frontend, mongoDB para la base de datos y telegram para la comunicación entre usuarios dando resultados positivos, favorables y fiables en el desarrollo del sistema. Este trabajo ha sido constatado acorde a sus resultados mostrados, sus pruebas realizadas a la aplicación, el tiempo de demora mínimo acorde a sus peticiones y el diseño de la aplicación.

A pesar de mencionar varios trabajos de investigación con arquitectura basada en microservicios se puede constatar que ninguno hace referencia a las tecnologías requeridas para esta investigación debido a que son tecnologías nuevas sin embargo con la profunda y delicada investigación se concuerda con las propuestas y objetivos presentados para realizar microservicios con eclipse Microprofile desplegados en Heroku sin ningún problema o complicación alguna, y por último teniendo en cuenta que pueden existir ciertas

modificaciones o adaptaciones más adelante que permitan tener un mejor rendimiento y diseño en la aplicación.

CAPÍTULO V

5. CONCLUSIONES

- Por medio de una entrevista directa al jefe de la empresa y el formato de requerimientos 830 se logró conocer los procesos sistemáticos en cuanto a las funciones que cumplía para llevar un control total de la empresa, lo cual contribuyó como sustento para el desarrollo del sistema de inventario basado en arquitectura de microservicios, elaborado en Eclipse Microprofile y desplegado en Heroku.
- Mediante la revisión documental, se pudo llegar a la conclusión de que esta tecnología a pesar de que permite realizar varios procesos sin tener que codificarlos, en la actualidad existe poca documentación por el momento por lo que la complejidad de desarrollo sea un poco mayor con relación a otras tecnologías.
- Para la base de datos escogimos una en la nube, aunque ésta es la mejor alternativa, nos hemos dado cuenta de que tiene un coste bastante alto para una pequeña aplicación debido a que cobran por uso y almacenamiento. No obstante, hemos podido analizar y concluir que para el inconveniente al que nos enfrentamos, la base de datos seleccionada es la ideal.
- Se logró llevar a cabo la finalidad de desarrollar una arquitectura capaz de recibir múltiples peticiones. Implementar la arquitectura de microservicios fue una labor de indagación debido a que cada una de las tecnologías usadas son bastante nuevas y es difícil descubrir información al respecto, sin embargo, gracias a la rigurosa investigación se pudo conseguir una aplicación limpia, escalable y fácil de mantener.
- Con la implementación de seguridad se obtuvo que no cualquier usuario pueda acceder a consultar toda la API REST sin limitación alguna, por lo que en la aplicación toda solicitud tiene que pasar por la puerta de seguridad con sus respetivas credenciales teniendo la conclusión de que las herramientas de seguridad son muy efectivas acorde al tipo de aplicación que se desee desarrollar.

CAPÍTULO VI

6. RECOMENDACIONES

- Se recomienda antes de implementar un sistema basado en arquitectura de microservicios, primero realizar un análisis profundo para poder determinar si la aplicación o el ecosistema tiene las características principales para adaptarse a la infraestructura de estar conformada por pequeños servicios. Debido a que esta arquitectura tiende a tener una complejidad muy superior a otras arquitecturas, sin embargo, el empleo de contenedores Docker permite estandarizar los servicios, de tal manera que puedan ser controlados mediante herramientas de monitoreo.
- Este trabajo de investigación deja como enseñanza lo importante que es estar en constante actualización acorde a nuevos temas de investigación, tecnologías, librerías, etc., es decir, las versiones más actuales en las funcionalidades de tecnologías deben tener un grado de importancia alto ya que pueden tener el objetivo de optimizar y automatizar aún más las aplicaciones web o sistemas informáticos.
- Los microservicios desarrollados mediante las herramientas definidas podrían requerir actualizaciones debido a que el framework lleva pocos años en el desarrollo y está en constante evolución, por lo que se requiere estar actualizados en las métricas que genere el framework a partir de cada actualización.
- Implementar, mejorar o complementar una interfaz gráfica para visualizar los datos de una forma mas interactiva, con nuevas funcionalidades sobre los datos desarrollados.

Referencias

[1] H. Liang, W. Chen, and K. Shi, “Cloud computing: Programming model and information exchange mechanism,” *Proceedings of the 2011 International Conference on Innovative Computing and Cloud Computing, ICCC’11*, pp. 10–12, 2011, doi: 10.1145/2071639.2071642.

[2] M. Almubaddel and A. M. Elmogy, “Cloud computing antecedents, challenges, and directions,” *ACM International Conference Proceeding Series*, vol. 22-23-Marc, 2016, doi: 10.1145/2896387.2896401.

[3] E. Cai, G. Kwan, P. Roubatsis, and Y.-K. Chang, “Building Microservices in a Cloud-Native World Using Eclipse Microprofile and Open Liberty,” *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*, pp. 350–353, 2018.

[4] R. Heinrich *et al.*, “Performance Engineering for Microservices,” pp. 223–226, 2017, doi: 10.1145/3053600.3053653.

[5] Z. Ren *et al.*, “Migrating web applications from monolithic structure to microservices architecture,” *ACM International Conference Proceeding Series*, 2018, doi: 10.1145/3275219.3275230.

[6] A. Jindal, V. Podolskiy, and M. Gerndt, “Performance modeling for cloud microservice applications,” *ICPE 2019 - Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, pp. 25–32, 2019, doi: 10.1145/3297663.3310309.

[7] C. Guzmán Alpízar, “Comunicación de microservicios usando colas de mensajes,” Universidad Politécnica de Madrid, 2019.

[8] A. Saransing, “Análisis de rendimiento entre una arquitectura monolítica y una arquitectura de microservicios - tecnología basada en contenedores,” *Universidad Técnica del Norte*, vol. 2, pp. 227–249, 2018.

[9] M. Williams, *A Quick Start Guide to Cloud Computing: Moving Your Business Into the Cloud*. 2010.

[10] G. Reese, *Cloud Application Architectures: Building Applications and Infrastructure in the Cloud*. O’Reilly Media, Inc., 2009.

[11] J. N. O.S. and S. Mary Saira Bhanu, “A Survey on Code Injection Attacks in Mobile Cloud Computing Environment,” in *2018 8th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, 2018, pp. 1–6. doi: 10.1109/CONFLUENCE.2018.8443032.

[12] C. Henriquez, J. F. Del Vecchio, and F. J. Paternina, “La computación en la nube: un modelo para el desarrollo de las empresas,” *Prospectiva*, vol. 13, no. 2, p. 81, 2015, doi: 10.15665/rp.v13i2.490.

[13] José Manuel and Navarro Arévalo, “Cloud Computing: Fundamentos, diseño y arquitectura aplicados a un caso de estudio Máster Oficial en Tecnologías de la Información y Sistemas Informáticos TESIS FIN DE MASTER Cloud Computing: fundamentos, diseño y arquitectura aplicados a un caso de estud,” *Tesis*, pp. 1–78, 2017.

[14] P. Mell and T. Grance, “The NIST definition of cloud computing,” in *Cloud Computing and Government: Background, Benefits, Risks*, Gaithersburg, MD, 2011, pp. 171–173. doi: 10.1016/b978-0-12-804018-8.15003-x.

[15] Instituto Nacional de Ciberseguridad, *Guía Cloud Computing*. 2017.

[16] J. Gao, X. Bai, W. Tsai, and T. Uehara, “Testing as a Service (TaaS) on Clouds,” in *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, 2013, pp. 212–223. doi: 10.1109/SOSE.2013.66.

[17] B. Lozano, *Cloud Computing Bible*. 2010.

[18] B. A. León-Velandia and M. A. Rosero-Muñoz, “Recomendaciones para contratar servicios en la ‘nube,’” *Revista Facultad De Ingeniería*, vol. 23, no. 37, p. 93, 2014, doi: 10.19053/01211129.2794.

[19] E. V. Alberto Ureña, Annie Ferrari, David Blanco, “Cloud Computing - Retos y Oportunidades,” *International Journal of Management & Information Systems*, vol. 16, no. 4, pp. 317–324, 2012.

[20] A. Gonzalez, “Gestión de proyectos de vinculación mediante un aplicativo web con mensajería instantánea y arquitectura de servicios,” *Pontificia Universidad Católica del Ecuador Sede Esmeraldas*, vol. 8, no. 5, p. 67, 2019.

[21] M. Fussell, “¿Por qué usar un enfoque de microservicios para crear aplicaciones?,” vol. 53, no. 9, pp. 1689–1699, 2017, doi: 10.1017/CBO9781107415324.004.

[22]. NET Foundation, “Communication in a microservice architecture,” 2020. https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture

[23]. NET Foundation, “Comunicación asincrónica basada en mensajes,” 2020. https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/asynchronous-message-based-communication

[24] E. Albertos Gómez, “Arquitecturas software para microservicios: una revisión sistemática de la literatura,” 2018.

[25] J. López, “Arquitectura de software basada en microservicios para desarrollo de aplicaciones web de la asamblea nacional,” *Universidad Técnica del Norte*, p. 162, 2017.

[26] E. E. Paredes Lozano, “Diseño de un prototipo de una arquitectura basada en microservicios para la integración de aplicaciones web altamente transaccionales. Caso entidades financieras,” *Universidad Politécnica Salesiana Sede Quito*, p. 272, 2013.

[27] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, and Y. Al-Hammadi, “The evolution of distributed systems towards microservices architecture,” *2016 11th International Conference for Internet Technology and Secured Transactions, ICITST 2016*, pp. 318–325, 2017, doi: 10.1109/ICITST.2016.7856721.

[28] C. P. Moreno, “Diseño e implementación de un microservicio con Spring,” 2019.

[29] E. Gamma, *Patrones de diseño: elementos de software orientado a objetos reutilizable*. Pearson Educación, 2002.

[30] D. Verdier and G. Rodriguez, “Implementación de Patrones de Microservicios,” pp. 5–129, 2020.

[31] E. Lavieri, *Hands-On Design Patterns*.

[32] R. Rocha and J. Purificação, *Java EE 8 Design Patterns and Best Practices*. 2020.

[33] S. Stark and P. I. Silva, “Eclipse MicroProfile Interoperable JWT RBAC,” p. 1 29, 2017.

[34] M. Jones, J. Bradley, and N. Sakimura, “RFC 7519: Json web token (JWT),” *Internet Engineering Task Force (IETF)*. p. 30, 2015.

[35] Y. Wilson and A. Hingnikar, “OpenID Connect,” in *Solving Identity Management in Modern Applications*, 2019, pp. 77–97. doi: 10.1007/978-1-4842-5095-2_6.

[36] N. Goldschlag, J. D. Kim, and M. Kristin, “[RFC 6749] The OAuth 2.0 Authorization Framework,” *Journal of Chemical Information and Modeling*, vol. 53, no. 9, pp. 1689–1699, 2019.

[37] L. Fugaro and M. Vocale, *Microservices with Jakarta EE*.

[38] “Getting Started with a Microprofile Project | Engineering Education (EngEd) Program | Section.” https://www.section.io/engineering-education/getting-hands-on-a-microprofile-project/ (accessed Aug. 01, 2022).

[39] J. Lindenbaum, A. Wiggins, and O. Henry, “Conoce más sobre Heroku,” 2007. https://www.heroku.com/about

[40] P. Danielsson, T. Postema, and H. Munir, “Heroku-based innovative platform for web-based deployment in product development at axis,” *IEEE Access*, vol. 9, no. January, pp. 10805–10819, 2021, doi: 10.1109/ACCESS.2021.3050255.

[41] B. H. Lee, E. K. Dewi, and M. F. Wajdi, “Data security in cloud computing using AES under HEROKU cloud,” *2018 27th Wireless and Optical Communication Conference, WOCC 2018*, no. April, pp. 1–5, 2018, doi: 10.1109/WOCC.2018.8372705.

[42] W. Rzasa, “Predicting performance in a paas environment: A case study for a web application,” *Computer Science*, vol. 18, no. 1, pp. 21–39, 2017, doi: 10.7494/csci.2017.18.1.21.

[43] L. Stewart, “Desarrollo front-end vs back-end: ¿Por dónde empezar?,” *Course Report*, 2020. https://www.coursereport.com/blog/front-end-development-vs-back-end-development-where-to-start#BackEndDevelopment

[44] J. L. P. LAJE, “Estudio Comparativo Entre Una Arquitectura Con Microservicios Y Contenedores Dockers Y Una Arquitectura Tradicional (Monolítica) Con Comprobación Aplicativa,” 2018.

[45] P. E. De la Cruz Vélez de Villa, M. H. Espinoza Ramirez, and O. Cuba Estrella, “Propuesta de arquitectura de microservicios, metodología Scrum para una aplicación móvil de control académico: Caso Escuela Profesional de Obstetricia de la UNMSM,” *Hamut’Ay*, vol. 6, no. 2, pp. 141–158, 2019, doi: 10.21503/hamu.v6i2.1781.

[46] R. Navarro and R. Cabrera, “Aplicación basada en Microservicios,” p. 153, 2020.

[47] A. Jindal *et al.*, “Increasing the dependability of IoT middleware with cloud computing and microservices,” *UCC 2017 - Proceedings of the10th International Conference on Utility and Cloud Computing*, vol. 2016, pp. 137–138, 2017, doi: 10.1109/INVENTIVE.2016.7830157.

[48] A. N. Ecuador, *Código Orgánico Integral Penal- Ley 0*. 2014, p. 144.

[49] D. P. Ecuador, *Utilizacion de software libre en la administracion publica*, vol. 25-abr.20. 2011, pp. 1–2.

[50] C. de E. S. Ecuador, *Ley Organica De Educacion Superior, LOES*. 2018, pp. 1–58.

[51] P. I. R. O. N. 320 Ecuador, *Registro Oficial No 320 Ley de Propiedad Intelectual*, no. 320. 2015, p. 92.

[52] A. N. del Ecuador, “Ley orgánica de emprendimiento e innovación,” no. 151, pp. 1–49, 2020.

[53] Asamblea Nacional del Ecuador, “Ley Organica de Telecomunicaciones,” p. 55, 2015.

[54] D. Causas, “Definición de las variables, enfoque y tipo de investigación,” *Universidad Nacional Abierta y a Distancia (UNAD)*, pp. 1–11, 2005.

ANEXOS

Anexo 1. Entrevista al jefe de la empresa Fujifilm

1) ¿En su empresa cuenta con un sistema de inventario informático?

Si [ ]

No [**X**]

2) En caso de adquirir uno, ¿quiénes serían los encargados de administrar el sistema?

Administrador [X]

Supervisor [ ]

Empleados [ ]

3) ¿De qué arquitectura preferiría que sea desarrollado el sistema? ¿Arquitectura monolítica o arquitectura de microservicios, y por qué?

Arquitectura monolítica [ ]

Arquitectura de microservicios [X]

4) ¿Qué procesos sistematizados cumple o realiza su empresa?

"Por el momento solo poseemos un software adquirido para la impresión de fotografías, el control de inventario lo llevamos a mano."

5) ¿Cuántos productos posee o maneja en su empresa?

"Manejamos pocos productos debido a que nos enfocamos en la impresión de fotografías, tenemos productos como recuadros para fotografías, cámaras fotográficas, baterías, usb, tarjetas de memoria y papel fotográfico"

Anexo 2. Requerimientos Funcionales

Tabla 14. Requerimiento funcional 1

Identificación del requerimiento:	RF01
Nombre del Requerimiento:	Autentificación de Usuario.
Características:	Los usuarios deberán identificarse para acceder a cualquier parte del sistema.
Descripción del requerimiento:	El sistema podrá ser consultado por cualquier usuario dependiendo del módulo en el cual se encuentre y su nivel de accesibilidad.
Requerimiento NO funcional:	• RNF01 • RNF02 • RNF05 • RNF08
Prioridad del requerimiento: Alta	

Tabla 15. Requerimiento funciona 2

Identificación del requerimiento:	RF02
Nombre del Requerimiento:	Registrar Usuarios.
Características:	Los usuarios deberán registrarse en el sistema para acceder a cualquier parte del sistema.
Descripción del requerimiento:	El sistema permitirá al usuario (personal y administrador) registrarse. El usuario debe suministrar datos como: CI, Nombre, Apellido, E-mail, Usuario y Contraseña.
Requerimiento NO funcional:	• RNF01 • RNF02 • RNF05 • RNF08
Prioridad del requerimiento: Alta	

Tabla 16. Requerimiento funcional 3

Identificación del requerimiento:	RF03
Nombre del Requerimiento:	Consultar Información.
Características:	El sistema ofrecerá al usuario información general acerca del inventario e información que posee la empresa.
Descripción del requerimiento:	Muestra información general sobre todos los productos que posee la empresa, que categoría pertenecen, y el tiempo de ingreso en stock.
Requerimiento NO funcional:	• RNF01 • RNF02
Prioridad del requerimiento: Alta	

Tabla 17. Requerimiento funcional 4

Identificación del requerimiento:	RF04
Nombre del Requerimiento:	Modificar.
Características:	El sistema permitirá al administrador modificar información y usuarios registrados.
Descripción del requerimiento:	Permite al administrador modificar datos de los usuarios y productos.
Requerimiento NO funcional:	• RNF01 • RNF02 • RNF05
Prioridad del requerimiento: Alta	

Tabla 18. Requerimiento funcional 5

Identificación del requerimiento:	RF05
Nombre del Requerimiento:	Gestión del Sistema de Inventario.
Características:	Permite gestionar información referente al Sistema de Inventario.
Descripción del requerimiento:	**Crear Categorías:** Permite al personal una vez que haya accedido con su cuenta, crear categorías de productos y suministrar información de los productos en inventario.
Requerimiento NO funcional:	• RNF01 • RNF02 • RNF05 • RNF06 • RNF07 • RNF08
Prioridad del requerimiento: Alta	

Tabla 19. Requerimiento funcional 6

Identificación del requerimiento:	RF06
Nombre del Requerimiento:	Gestión del Sistema de Inventario.
Características:	Permite gestionar información referente al Sistema de Inventario.
Descripción del requerimiento:	**Registrar Producto** El producto deberá ser suministrado con su código, precio y cantidad disponible en el inventario.
Requerimiento NO funcional:	• RNF01 • RNF02 • RNF05 • RNF06 • RNF07 • RNF08
Prioridad del requerimiento: Alta	

Tabla 20. Requerimiento funcional 7

Identificación del requerimiento:	RF07
Nombre del Requerimiento:	Gestión del Sistema de Inventario.
Características:	Permite gestionar información referente al Sistema de Inventario.
Descripción del requerimiento:	**Consultar Producto:** permite al personar ver información del inventario referente a los productos y ventas realizadas en un determinado tiempo.
Requerimiento NO funcional:	• RNF01 • RNF02 • RNF05 • RNF06 • RNF07 • RNF08
Prioridad del requerimiento:	Alta

Tabla 21. Requerimiento funcional 8

Identificación del requerimiento:	RF08
Nombre del Requerimiento:	Gestión del Sistema de Inventario.
Características:	Permite gestionar información referente al Sistema de Inventario.
Descripción del requerimiento:	**Modificar Producto:** permite al personal modificar información del inventario referente a los productos. y ventas realizadas en un determinado tiempo.
Requerimiento NO funcional:	• RNF01 • RNF02 • RNF05 • RNF06 • RNF07 • RNF08
Prioridad del requerimiento:	Alta

Tabla 22. Requerimiento funcional 9

Identificación del requerimiento:	RF09
Nombre del Requerimiento:	Gestión del Sistema de Inventario.
Características:	Permite gestionar información referente al Sistema de Inventario.
Descripción del requerimiento:	**Eliminar Producto:** permite al personal eliminar información del inventario referente a los productos y ventas realizadas en un determinado tiempo.
Requerimiento NO funcional:	• RNF01 • RNF02 • RNF05 • RNF06 • RNF07 • RNF08
Prioridad del requerimiento:	Alta

Anexo 3. Requerimientos No Funcionales

Tabla 23. Requerimiento no funcional 1

Identificación del requerimiento:	RNF01
Nombre del Requerimiento:	Interfaz del sistema.
Características:	El sistema tendrá una interfaz sencilla.
Descripción del requerimiento:	El sistema debe tener una interfaz de uso intuitiva y didáctica.
Prioridad del requerimiento: Alta	

Tabla 21. Requerimiento no funcional 2

Identificación del requerimiento:	RNF02
Nombre del Requerimiento:	Mantenimiento.
Características:	Los microservicios tendrán una guía de cómo fueron implementados, sus dependencias y codificación.
Descripción del requerimiento:	El sistema debe disponer de una documentación fácilmente actualizable que permita realizar operaciones de mantenimiento con el menor esfuerzo posible.
Prioridad del requerimiento: Alta	

Tabla 24. Requerimiento no funcional 3

Identificación del requerimiento:	RNF03
Nombre del Requerimiento:	Diseño de la interfaz a la característica de la web.
Características:	El sistema deberá de tener una interfaz de usuario, teniendo en cuenta las características de la web de la institución.
Descripción del requerimiento:	La interfaz de usuario debe ajustarse a las características de la web de la institución, dentro de la cual estará incorporado el sistema de gestión de procesos y el inventario.
Prioridad del requerimiento: Alta	

Tabla 25. Requerimiento no funcional 4

Identificación del requerimiento:	RNF04
Nombre del Requerimiento:	Desempeño
Características:	El sistema garantizara a los usuarios un desempeño en cuanto a los datos almacenado en el sistema ofreciéndole una confiabilidad a esta misma.
Descripción del requerimiento:	Garantizar el desempeño del sistema informático a los diferentes usuarios. En este sentido la información almacenada o registros realizados podrán ser consultados y actualizados permanente y simultáneamente, sin que se afecte el tiempo de respuesta.
Prioridad del requerimiento: Alta	

Tabla 26. Requerimiento no funcional 5

Identificación del requerimiento:	RNF05
Nombre del Requerimiento:	Nivel de Usuario
Características:	Garantizara al usuario el acceso de información de acuerdo al rol que posea.
Descripción del requerimiento:	Control al acceso de información acorde al rol que cumpla el usuario.
Prioridad del requerimiento: Alta	

Tabla 27. Requerimiento no funcional 6

Identificación del requerimiento:	RNF06
Nombre del Requerimiento:	Confiabilidad continúa de los microservicios.
Características:	El sistema tendrá que estar en funcionamiento constantemente.
Descripción del requerimiento:	La disponibilidad de los microservicios debe ser continua con un nivel de servicio eficiente.
Prioridad del requerimiento: Alta	

Anexo 4. Código de la aplicación

Archivo pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.pucese</groupId>
  <artifactId>producto</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
  <properties>
    <maven.compiler.target>1.8</maven.compiler.target>
    <failOnMissingWebXml>false</failOnMissingWebXml>
    <maven.compiler.source>1.8</maven.compiler.source>
    <tomee.version>8.0.0-M3</tomee.version>
    <final.name>producto</final.name>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.eclipse.microprofile</groupId>
      <artifactId>microprofile</artifactId>
      <version>2.1</version>
      <type>pom</type>
      <scope>provided</scope>
    </dependency>
    <dependency>
       <groupId>com.kumuluz.ee</groupId>
       <artifactId>kumuluzee-cdi-weld</artifactId>
    </dependency>
    <dependency>
       <groupId>com.kumuluz.ee.security</groupId>
       <artifactId>kumuluzee-security-keycloak</artifactId>
       <version>${kumuluzee-security.version}</version>
    </dependency>
    <dependency>
         <groupId>org.apache.cxf</groupId>
         <artifactId>cxf-rt-rs-client</artifactId>
         <version>3.1.13</version>
          <scope>provided</scope>
    </dependency>
  </dependencies>
  <build>
    <finalName>producto</finalName>
  </build>
  <profiles>
    <profile>
      <id>tomee</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <build>
        <plugins>
          <plugin>
            <groupId>org.apache.tomee.maven</groupId>
            <artifactId>tomee-maven-plugin</artifactId>
            <version>${tomee.version}</version>
            <executions>
              <execution>
                <id>executable-jar</id>
                <phase>package</phase>
                <goals>
                  <goal>exec</goal>
                </goals>
              </execution>
```

```
            </executions>
            <configuration>
              <context>ROOT</context>
              <tomeeClassifier>microprofile</tomeeClassifier>
              <tomeeHttpPort>8080</tomeeHttpPort>
              <tomeeShutdownPort>8005</tomeeShutdownPort>
              <tomeeAjpPort>8009</tomeeAjpPort>
            </configuration>
          </plugin>
        </plugins>
      </build>
    </profile>
  </profiles>
</project>
```

MICROSERVICIO DE SEGURIDAD E INICIO DE SESIÓN

Archivo resources/config.yaml

```
kumuluzee:
  security:
      keycloak:
        json: '{"realm": "customers-realm",
                "bearer-only": true,
                "auth-server-url": "http://localhost:8080/auth",
                "ssl-required": "external",
                "resource": "customers-api"}'
```

Archivo principal del servicio REST

```
@DeclareRoles({"user", "admin"})
@ApplicationPath("/login")
public class CustomerApplication extends Application {
}
```

Restricciones de seguridad recurso JAX-RS

```
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
@Path("customers")
public class CustomerResource {

    @GET
    @PermitAll
    public Response getAllCustomers() {
    }
    @GET
    @Path("{customerId}")
    public Response getCustomer(@PathParam("customerId") String customerId) {
    }
    @POST
    @RolesAllowed("user")
    public Response addNewCustomer(Customer customer) {
    }
    @DELETE
    @Path("{customerId}")
    @RolesAllowed("user")
    public Response deleteCustomer(@PathParam("customerId") String customerId) {
    }
}
```

MICROSERVICIO DEL CRUD Y CONEXIÓN A LA BASE DE DATOS

Entities/producto.java

```
package com.pucese.producto.entities;

import java.io.Serializable;
import javax.validation.constraints.Size;
import java.util.Objects;
import javax.persistence.*;


@Entity
@Table(name = "producto")
@NamedQueries({
        @NamedQuery(name = "Producto.findAll", query = "SELECT p
FROM Producto p")
})
public class Producto implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
       @GeneratedValue(strategy = GenerationType.IDENTITY)
       @Column(name = "Id")
       private Long Id;

       @Column(name = "nombre", length = 50)
       private String nombre;

       @Column(name = "procantidadduct_stock")
       private int cantidad;

       @Column(name = "disponible")
       private boolean disponible;

       @Column(name = "precio")
       private double precio;

       public Producto() {

       }

       public Producto(String nombre, int cantidad, double precio) {
           this.nombre = nombre;
           this.cantidad = cantidad;
           this.precio = precio;
           this.disponible = true;
       }


       public Long getId() {
         return Id;
      }

      public void setId(Long id) {
         Id = id;
```

```
}

public String getNombre() {
   return nombre;
}

public void setNombre(String nombre) {
   this.nombre = nombre;
}

public int getCantidad() {
   return cantidad;
}

public void setCantidad(int cantidad) {
   this.cantidad = cantidad;
}

public boolean isDisponible() {
   return disponible;
}

public void setDisponible(boolean disponible) {
   this.disponible = disponible;
}

public double getPrecio() {
   return precio;
}

public void setPrecio(double precio) {
   this.precio = precio;
}

@Override
 public int hashCode() {
     int hash = 5;
     hash = 71 * hash + Objects.hashCode(this.Id);
     return hash;
 }

 @Override
 public boolean equals(Object obj) {
     if (this == obj) {
         return true;
     }
     if (obj == null) {
         return false;
     }
     if (getClass() != obj.getClass()) {
         return false;
     }
     final Producto other = (Producto) obj;
     return Objects.equals(this.Id, other.Id);
 }

 @Override
```

```
       public String toString() {
           return "Product{" + "productId=" + disponible + ",
nombre=" + nombre + ", cantidad=" + cantidad + ", disponible=" +
disponible + ", precio=" + precio + '}';
       }

   }
```

Repositories/ProductoRepository.java

```
package com.pucese.producto.repositories;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import com.pucese.producto.entities.Producto;

import java.util.List;

@Stateless
public class ProductoRepository {

    @PersistenceContext(unitName = "myBlog_PU")
    EntityManager em;

    public List getAllProducto() {
        return em.createNamedQuery("Producto.findAll",
Producto.class).getResultList();
    }

    public Producto findById(Long id) {
        return em.find(Producto.class, id);
    }

    public Producto create(Producto producto) {
        em.persist(producto);
        return producto;
    }

    public void update(Producto producto) {
        em.merge(producto);
    }

    public void delete(Producto producto) {
        if (!em.contains(producto)) {
          producto = em.merge(producto);
        }

        em.remove(producto);
    }
}
```

Resource/ProductoResource.java

```
package com.pucese.producto.resources;

import javax.enterprise.context.RequestScoped;
import javax.inject.Inject;
import javax.ws.rs.*;
import javax.ws.rs.core.*;

import com.pucese.producto.entities.Producto;
import com.pucese.producto.repositories.ProductoRepository;

@RequestScoped
@Path("producto")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)

public class ProductoResource {

    @Inject//conexion con la capa de datos
    ProductoRepository productoRepository;

    @GET//metodo para mostrar todos
    public Response getAllProducto() {
        return
Response.ok().entity(productoRepository.getAllProducto()).build();
    }
    @GET
    @Path("{id}")
    public Response getPostById(@PathParam("id") Long id) {
        return
Response.ok().entity(productoRepository.findById(id)).build();
    }
    @POST
    public Response create(Producto producto, @Context UriInfo
uriInfo) {
     Producto productoId = productoRepository.create(producto);
        UriBuilder builder = uriInfo.getAbsolutePathBuilder();
        builder.path(Long.toString(productoId.getId()));
        return Response.created(builder.build()).build();
    }
    @PUT
    @Path("{id}")
    public Response update(@PathParam("id") Long id, Producto
producto) {
     Producto updateProducto = productoRepository.findById(id);

     updateProducto.setNombre(producto.getNombre());
     updateProducto.setCantidad(producto.getCantidad());
     updateProducto.setPrecio(producto.getPrecio());
     updateProducto.setDisponible(producto.isDisponible());

        return Response.ok().entity(producto).build();
    }
```

```
    @DELETE
    @Path("{id}")
    public Response delete(@PathParam("id") Long id) {
        Producto producto = productoRepository.findById(id);
        productoRepository.delete(producto);
        return Response.noContent().build();
    }
}
```

ProductoRestApplication.java

```
package com.pucese.producto;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

/**
 *
 */
@ApplicationPath("/data")
public class ProductoRestApplication extends Application {
}
```